Stochastic Process Algebras

Jane Hillston and Mirco Tribastone.
LFCS,
University of Edinburgh

29th May 2007

Modelling with Markov chains

In his lecture on Monday Billy Stewart said that there are three steps to modelling with Markov chains:

Modelling with Markov chains

In his lecture on Monday Billy Stewart said that there are three steps to modelling with Markov chains:

1. Conceptualise your system as a Markov chain;

Modelling with Markov chains

In his lecture on Monday Billy Stewart said that there are three steps to modelling with Markov chains:

1. Conceptualise your system as a Markov chain;
2. Construct your Markov chain — construct an infinitesimal generator matrix Q.

Modelling with Markov chains

In his lecture on Monday Billy Stewart said that there are three steps to modelling with Markov chains:

1. Conceptualise your system as a Markov chain;
2. Construct your Markov chain — construct an infinitesimal generator matrix Q.
3. Solve your Markov chain to derive quantitative information about the system.

Modelling with Markov chains

In his lecture on Monday Billy Stewart said that there are three steps to modelling with Markov chains:

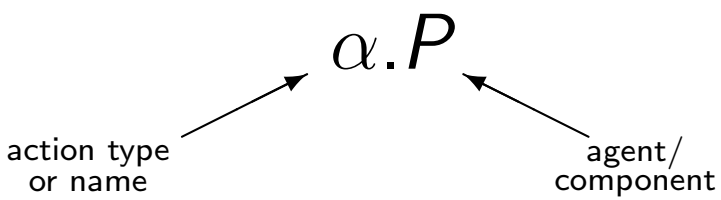
1. Conceptualise your system as a Markov chain;
2. Construct your Markov chain — construct an infinitesimal generator matrix Q.
3. Solve your Markov chain to derive quantitative information about the system.

Outline

Outline

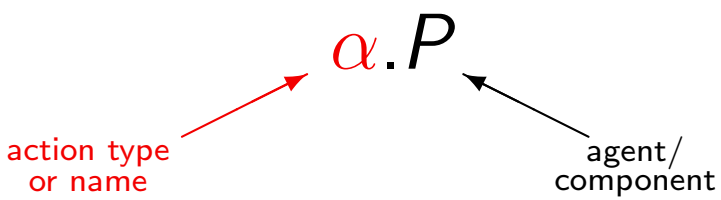
Process Algebra

- Models consist of agents which engage in actions.



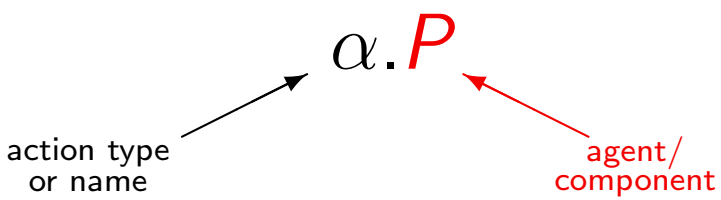
Process Algebra

- Models consist of agents which engage in actions.



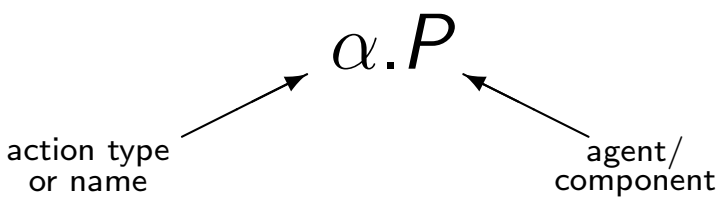
Process Algebra

- Models consist of agents which engage in actions.



Process Algebra

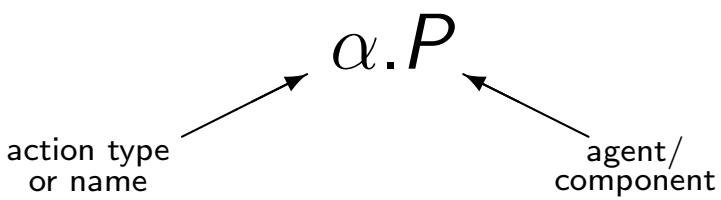
- Models consist of agents which engage in actions.



- The structured operational (interleaving) semantics of the language is used to generate a labelled transition system.

Process Algebra

- Models consist of agents which engage in actions.

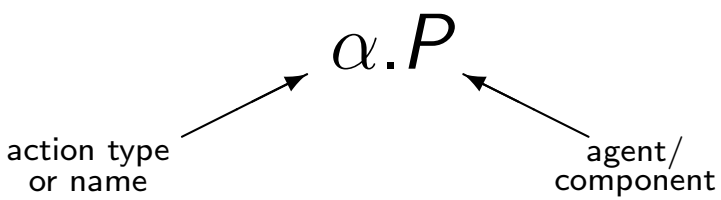


- The structured operational (interleaving) semantics of the language is used to generate a labelled transition system.

Process algebra model

Process Algebra

- Models consist of agents which engage in actions.

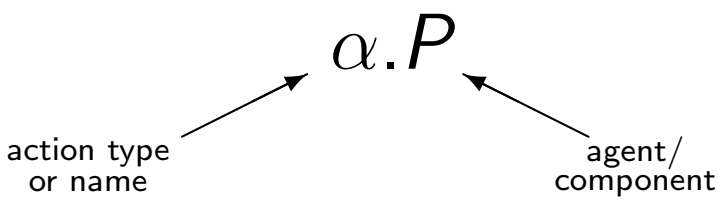


- The structured operational (interleaving) semantics of the language is used to generate a labelled transition system.

Process algebra model $\xrightarrow{\text{SOS rules}}$

Process Algebra

- Models consist of agents which engage in actions.



- The structured operational (interleaving) semantics of the language is used to generate a labelled transition system.

Process algebra model $\xrightarrow{\text{SOS rules}}$ Labelled transition system

Example

Consider a web server which offers html pages for download:

$$Server \stackrel{def}{=} get.download.rel.Server$$

Example

Consider a web server which offers html pages for download:

$$Server \stackrel{def}{=} get.download.rel.Server$$

Its clients might be web browsers, in a domain with a local cache of frequently requested pages. Thus any display request might result in an access to the server or in a page being loaded from the cache.

$$Browser \stackrel{def}{=} display.(cache.Browser + get.download.rel.Browser)$$

Example

Consider a web server which offers html pages for download:

$$Server \stackrel{def}{=} get.download.rel.Server$$

Its clients might be web browsers, in a domain with a local cache of frequently requested pages. Thus any display request might result in an access to the server or in a page being loaded from the cache.

$$Browser \stackrel{def}{=} display.(cache.Browser + get.download.rel.Browser)$$

A simple version of the Web can be considered to be the interaction of these components:

$$WEB \stackrel{def}{=} (Browser \parallel Browser) \mid Server$$

Dynamic behaviour

- The behaviour of a model is dictated by the semantic rules governing the combinators of the language.

Dynamic behaviour

- The behaviour of a model is dictated by the semantic rules governing the combinators of the language.
- The possible evolutions of a model are captured by applying these rules exhaustively, generating a labelled transition system.

Dynamic behaviour

- The behaviour of a model is dictated by the semantic rules governing the combinators of the language.
- The possible evolutions of a model are captured by applying these rules exhaustively, generating a labelled transition system.
- This can be viewed as a graph in which each node is a state of the model (comprised of the local states of each of the components) and the arcs represent the actions which can cause the move from one state to another.

Dynamic behaviour

- The behaviour of a model is dictated by the semantic rules governing the combinators of the language.
- The possible evolutions of a model are captured by applying these rules exhaustively, generating a labelled transition system.
- This can be viewed as a graph in which each node is a state of the model (comprised of the local states of each of the components) and the arcs represent the actions which can cause the move from one state to another.
- The language is also equipped with observational equivalence which can be used to compare models.

Dynamic behaviour

$$Browser \stackrel{def}{=} display.(cache.Browser \quad + \quad get.download.rel.Browser)$$

Dynamic behaviour

$$Browser \stackrel{def}{=} display.(cache.Browser \quad + \quad get.download.rel.Browser)$$

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}$$

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

$$\frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

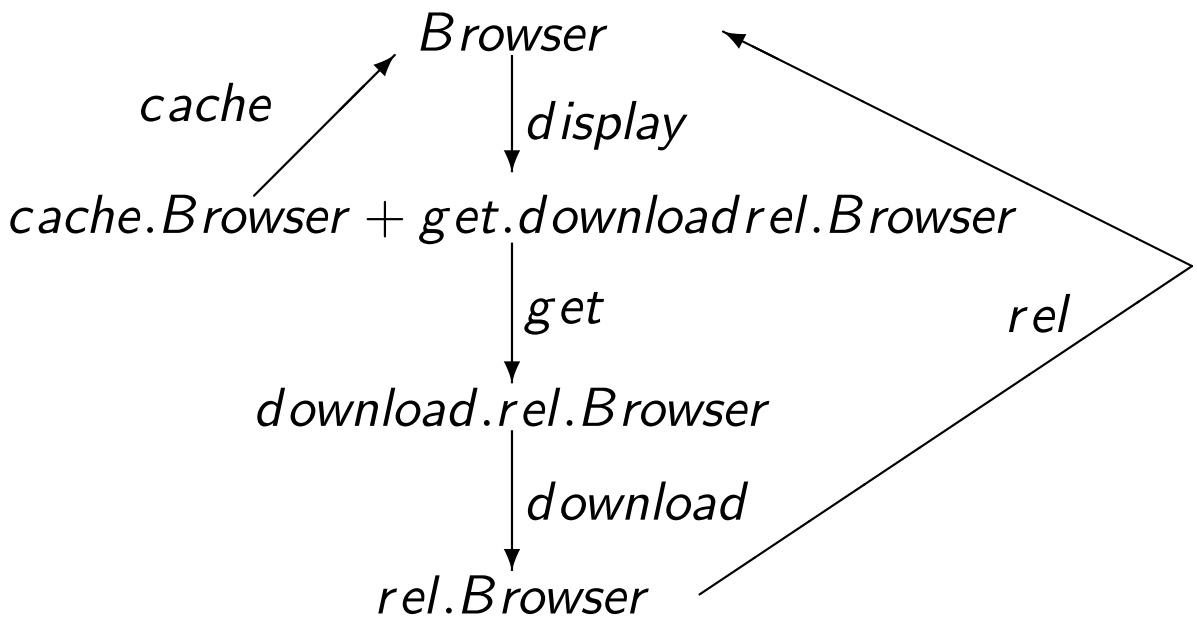
Dynamic behaviour

$$Browser \stackrel{def}{=} display.(cache.Browser \ + \ get.download.rel.Browser)$$

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}$$

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

$$\frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$



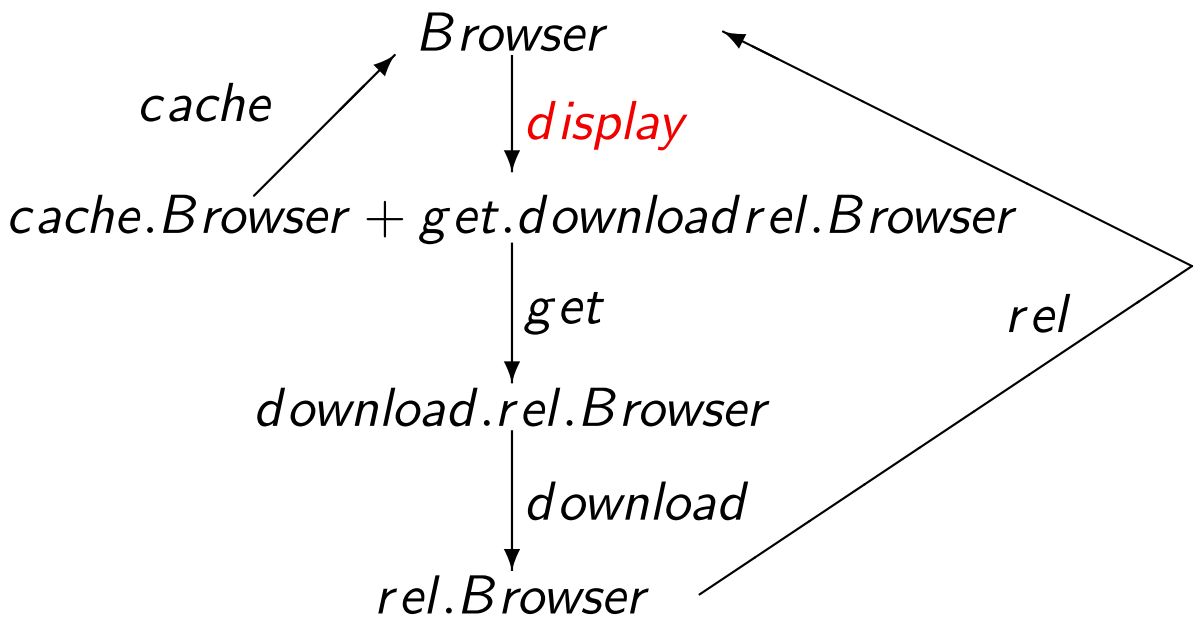
Dynamic behaviour

$$Browser \stackrel{def}{=} display.(cache.Browser \ + \ get.download.rel.Browser)$$

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}$$

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

$$\frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$



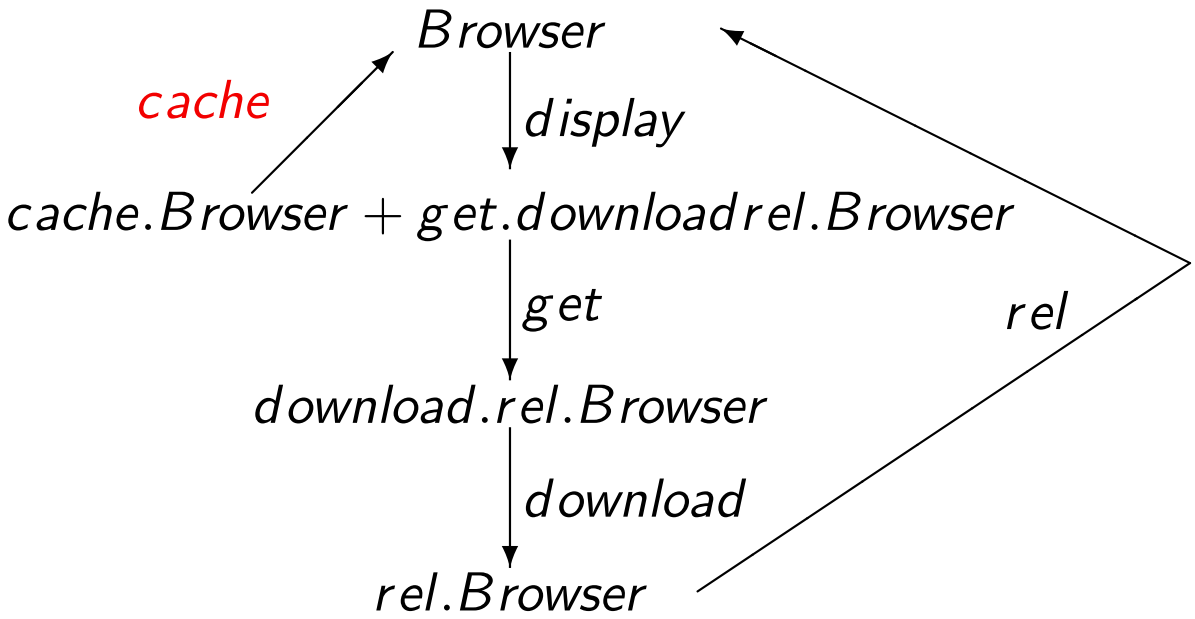
Dynamic behaviour

$$Browser \stackrel{def}{=} display.(cache.Browser + get.download.rel.Browser)$$

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}$$

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

$$\frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$



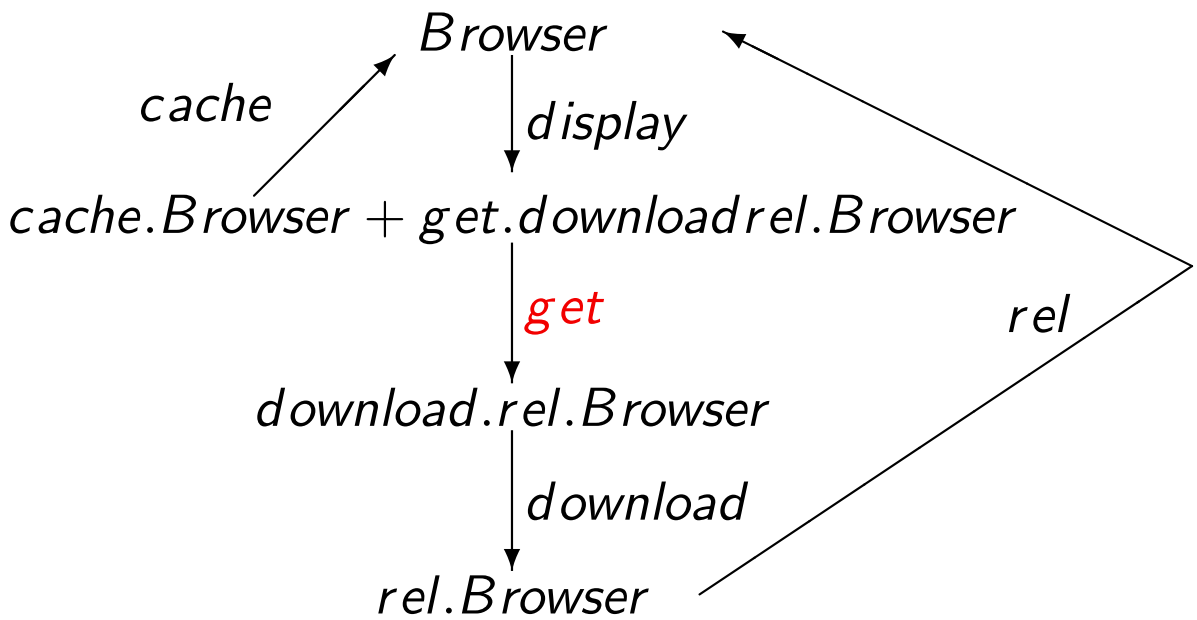
Dynamic behaviour

$$Browser \stackrel{def}{=} display.(cache.Browser \ + \ get.download.rel.Browser)$$

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}$$

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

$$\frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$



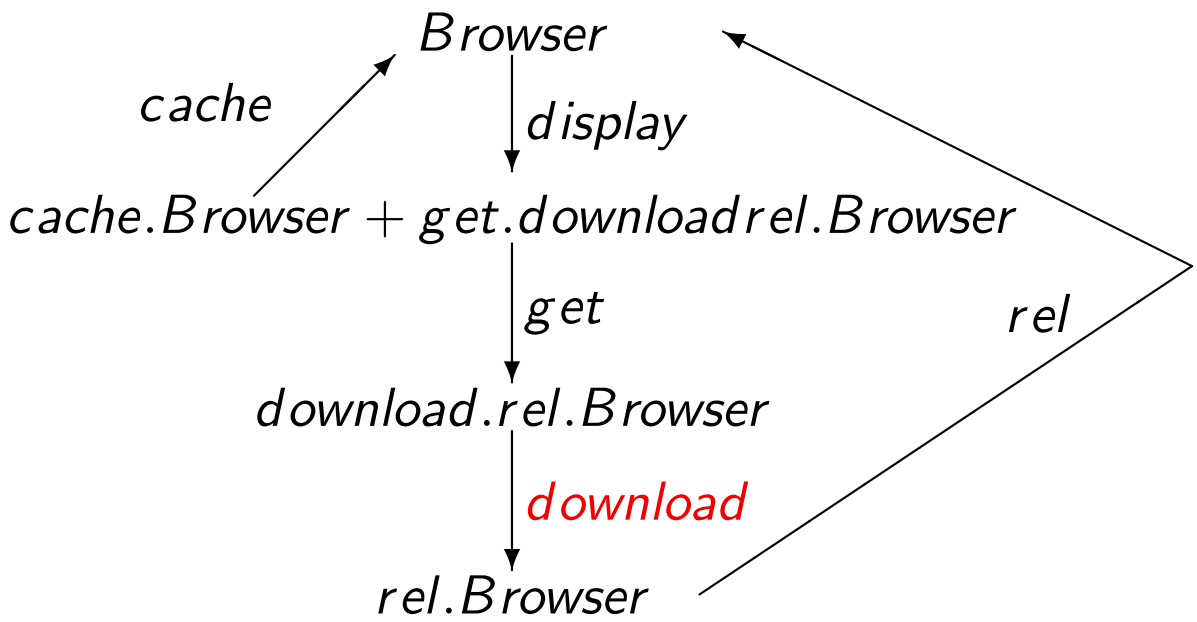
Dynamic behaviour

$$Browser \stackrel{def}{=} display.(cache.Browser \;+\; get.download.rel.Browser)$$

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}$$

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

$$\frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$



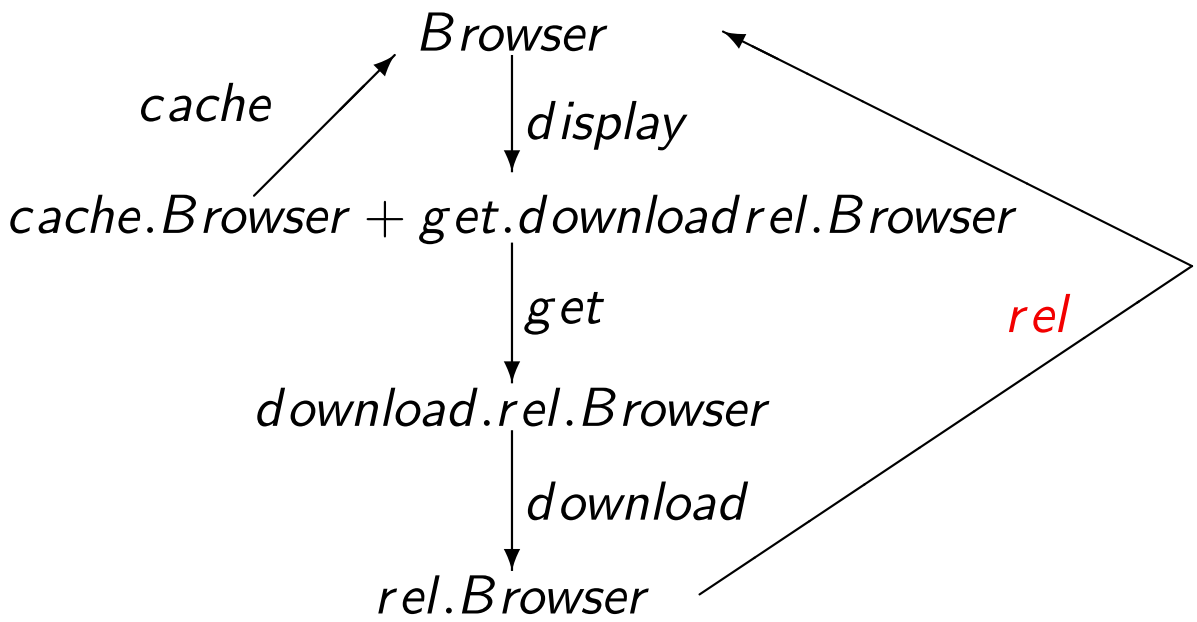
Dynamic behaviour

$$Browser \stackrel{def}{=} display.(cache.Browser \ + \ get.download.rel.Browser)$$

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}$$

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

$$\frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$



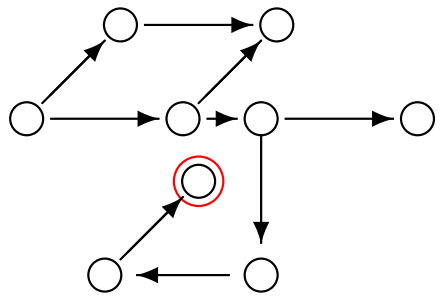
Qualitative Analysis

- The labelled transition system underlying a process algebra model can be used for functional verification e.g.: reachability analysis, specification matching and model checking.

Qualitative Analysis

- The labelled transition system underlying a process algebra model can be used for functional verification e.g.: reachability analysis, specification matching and model checking.

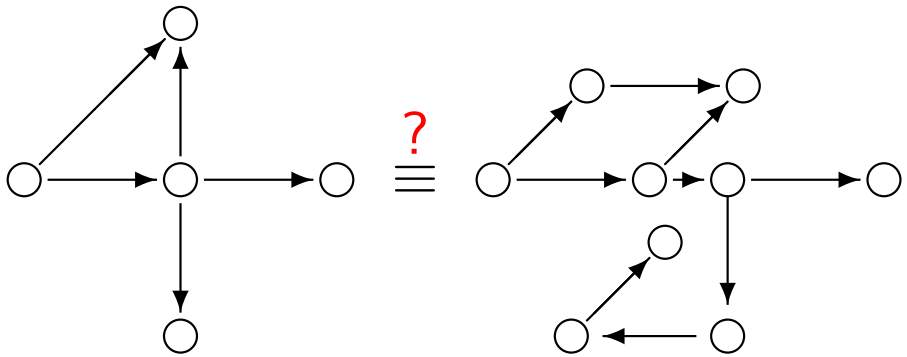
Will the system arrive in a particular state?



Qualitative Analysis

- The labelled transition system underlying a process algebra model can be used for functional verification e.g.: reachability analysis, specification matching and model checking.

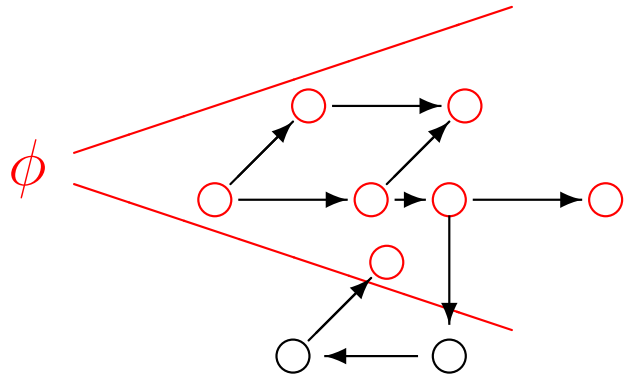
Does system behaviour match its specification?



Qualitative Analysis

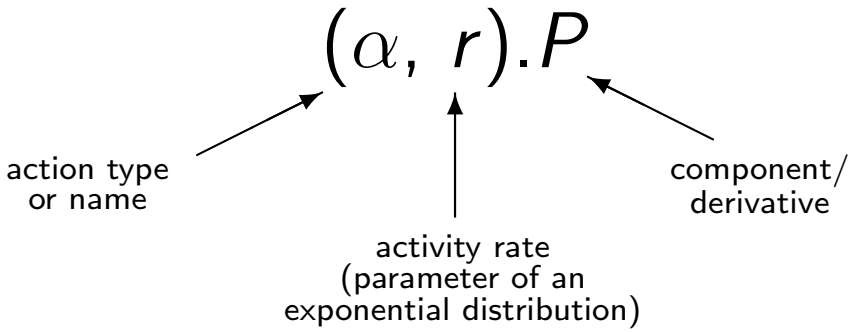
- The labelled transition system underlying a process algebra model can be used for functional verification e.g.: reachability analysis, specification matching and model checking.

Does a given property ϕ hold within the system?



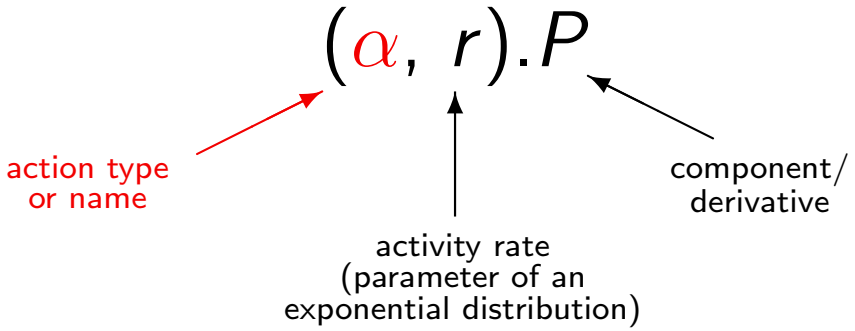
Performance Evaluation Process Algebra

- Models are constructed from components which engage in activities.



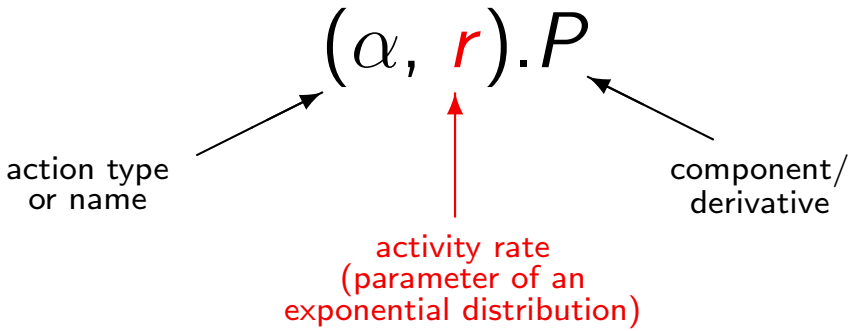
Performance Evaluation Process Algebra

- Models are constructed from components which engage in activities.



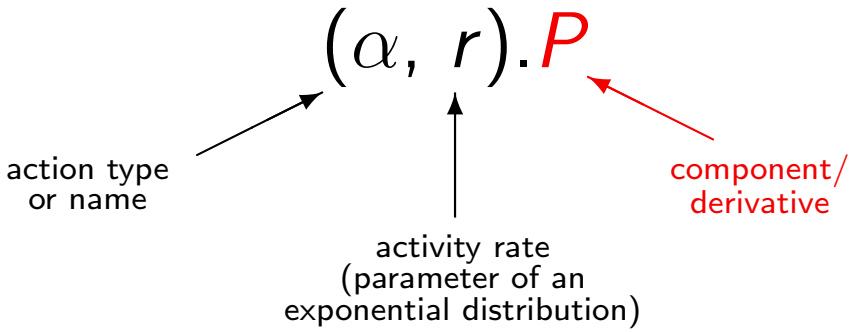
Performance Evaluation Process Algebra

- Models are constructed from components which engage in activities.



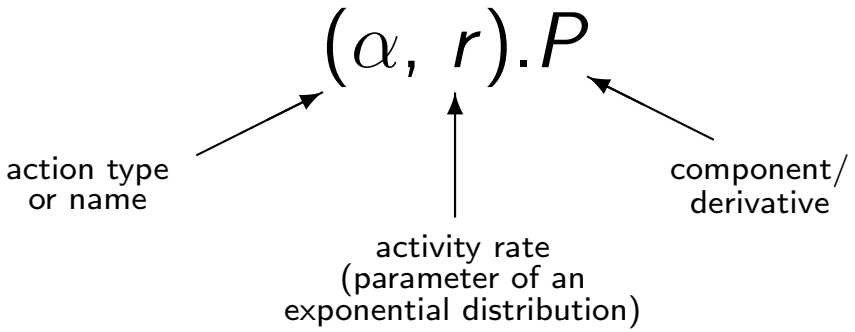
Performance Evaluation Process Algebra

- Models are constructed from components which engage in activities.



Performance Evaluation Process Algebra

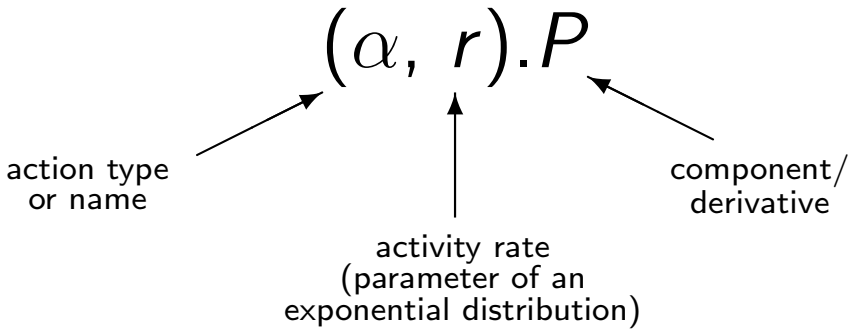
- Models are constructed from components which engage in activities.



- The language is used to generate a CTMC for performance modelling.

Performance Evaluation Process Algebra

- Models are constructed from components which engage in activities.

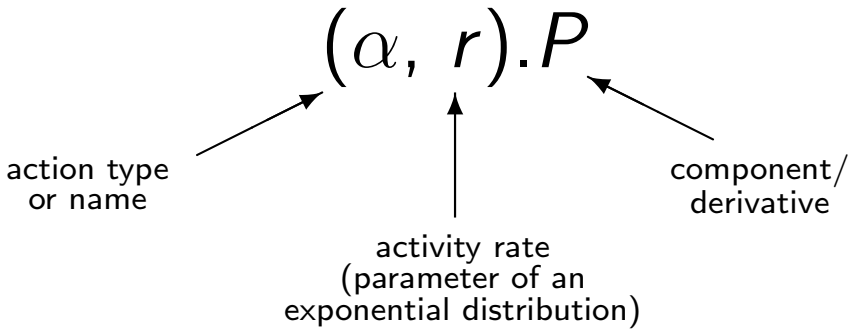


- The language is used to generate a CTMC for performance modelling.

PEPA
MODEL

Performance Evaluation Process Algebra

- Models are constructed from components which engage in activities.

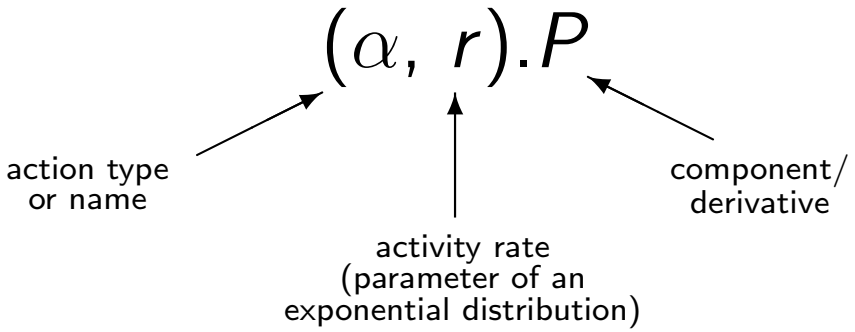


- The language is used to generate a CTMC for performance modelling.

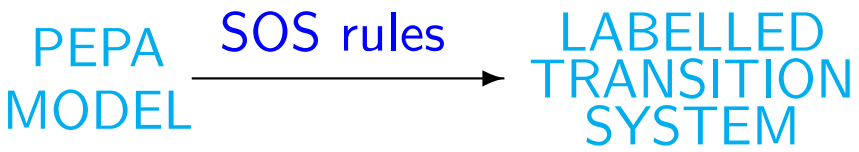
PEPA MODEL $\xrightarrow{\text{SOS rules}}$

Performance Evaluation Process Algebra

- Models are constructed from components which engage in activities.

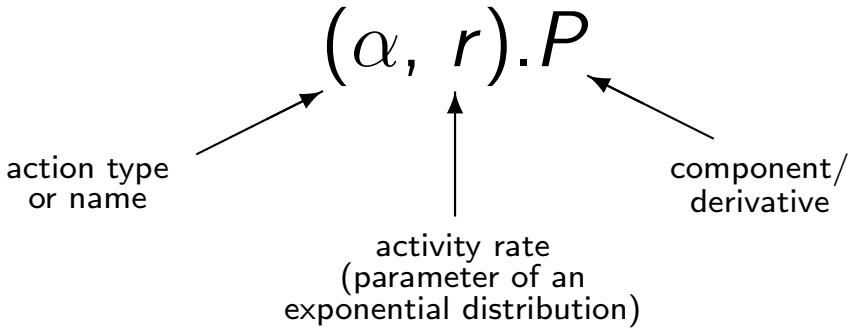


- The language is used to generate a CTMC for performance modelling.



Performance Evaluation Process Algebra

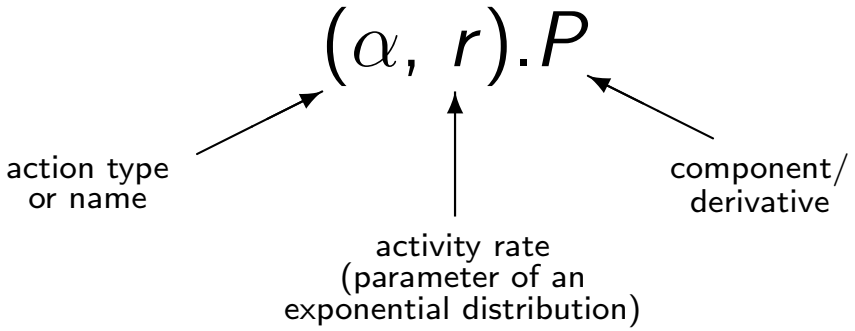
- Models are constructed from components which engage in activities.



- The language is used to generate a CTMC for performance modelling.

Performance Evaluation Process Algebra

- Models are constructed from components which engage in activities.



- The language is used to generate a CTMC for performance modelling.

PEPA

$$
\begin{aligned}
S &::= (\alpha, r).S \mid S + S \mid A \\
P &::= S \mid P \underset{L}{\bowtie} P \mid P/L
\end{aligned}
$$

PEPA

$$
\begin{aligned}
S &::= (\alpha, r).S \mid S + S \mid A \\
P &::= S \mid P \underset{L}{\bowtie} P \mid P/L
\end{aligned}
$$

PREFIX: $(\alpha, r).S$ designated first action

PEPA

$$
\begin{aligned}
S &::= (\alpha, r).S \mid S + S \mid A \\
P &::= S \mid P \underset{L}{\bowtie} P \mid P/L
\end{aligned}
$$

PREFIX:	$(\alpha, r).S$	designated first action
CHOICE:	$S + S$	competing components (race policy)

PEPA

$$
\begin{aligned}
S &::= (\alpha, r).S \mid S + S \mid A \\
P &::= S \mid P \underset{L}{\bowtie} P \mid P/L
\end{aligned}
$$

PREFIX:	$(\alpha, r).S$	designated first action
CHOICE:	$S + S$	competing components (race policy)
CONSTANT:	$A \stackrel{def}{=} S$	assigning names

PEPA

$$
\begin{aligned}
S &::= (\alpha, r).S \mid S + S \mid A \\
P &::= S \mid P \underset{L}{\bowtie} P \mid P/L
\end{aligned}
$$

PREFIX:	$(\alpha, r).S$	designated first action
CHOICE:	$S + S$	competing components (race policy)
CONSTANT:	$A \stackrel{def}{=} S$	assigning names
COOPERATION:	$P \underset{L}{\bowtie} P$	$\alpha \notin L$ concurrent activity (*individual actions*) $\alpha \in L$ cooperative activity (*shared actions*)

PEPA

$$
\begin{aligned}
S &::= (\alpha, r).S \mid S + S \mid A \\
P &::= S \mid P \underset{L}{\bowtie} P \mid P/L
\end{aligned}
$$

PREFIX:	$(\alpha, r).S$	designated first action
CHOICE:	$S + S$	competing components (race policy)
CONSTANT:	$A \stackrel{def}{=} S$	assigning names
COOPERATION:	$P \underset{L}{\bowtie} P$	$\alpha \notin L$ concurrent activity (*individual actions*) $\alpha \in L$ cooperative activity (*shared actions*)

PEPA

$$
\begin{aligned}
S &::= (\alpha, r).S \mid S + S \mid A \\
P &::= S \mid P \underset{L}{\bowtie} P \mid P/L
\end{aligned}
$$

PREFIX:	$(\alpha, r).S$	designated first action
CHOICE:	$S + S$	competing components (race policy)
CONSTANT:	$A \stackrel{def}{=} S$	assigning names
COOPERATION:	$P \underset{L}{\bowtie} P$	$\alpha \notin L$ concurrent activity (*individual actions*) $\alpha \in L$ cooperative activity (*shared actions*)
HIDING:	P/L	abstraction $\alpha \in L \Rightarrow \alpha \rightarrow \tau$

Example revisited

The behaviour of the server is the same but now quantitative information is recorded for each operation:

$$Server \stackrel{def}{=} (get, \top).(download, \mu).(rel, \top).Server$$

Example revisited

The behaviour of the server is the same but now quantitative information is recorded for each operation:

$$Server \stackrel{def}{=} (get, \top).(download, \mu).(rel, \top).Server$$

In addition to duration we also incorporate information about the relative frequencies of the different actions which take place after a display request:

$$Browser \stackrel{def}{=} (display, p_1\lambda).(cache, m).Browser + (display, p_2\lambda).(get, g).(download, \top).(rel, r).Browser$$

Example revisited

The behaviour of the server is the same but now quantitative information is recorded for each operation:

$$Server \stackrel{def}{=} (get, \top).(download, \mu).(rel, \top).Server$$

In addition to duration we also incorporate information about the relative frequencies of the different actions which take place after a display request:

$$\begin{aligned} Browser \stackrel{def}{=} & (display, p_1\lambda).(cache, m).Browser + \\ & (display, p_2\lambda).(get, g).(download, \top).(rel, r).Browser \end{aligned}$$

The configuration is recorded as before; using the PEPA cooperation the actions which must be shared are explicitly named:

$$WEB \stackrel{def}{=} (Browser \parallel Browser) \underset{L}{\bowtie} Server$$

$$L = \{get, download, rel\}$$

Example revisited

The behaviour of the server is the same but now quantitative information is recorded for each operation:

$$Server \stackrel{def}{=} (get, \top).(download, \mu).(rel, \top).Server$$

In addition to duration we also incorporate information about the relative frequencies of the different actions which take place after a display request:

$$\begin{aligned} Browser \stackrel{def}{=} & (display, p_1\lambda).(cache, m).Browser + \\ & (display, p_2\lambda).(get, g).(download, \top).(rel, r).Browser \end{aligned}$$

The configuration is recorded as before; using the PEPA cooperation the actions which must be shared are explicitly named:

$$WEB \stackrel{def}{=} (Browser \parallel Browser) \underset{L}{\bowtie} Server$$

$$L = \{get, download, rel\}$$

Integrated analysis

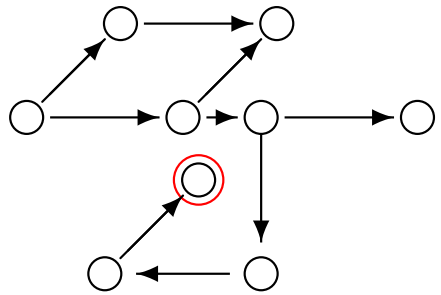
- Qualitative verification can now be complemented by quantitative verification:

Integrated analysis

- Qualitative verification can now be complemented by quantitative verification:

Reachability analysis

How long will it take for the system to arrive in a particular state?

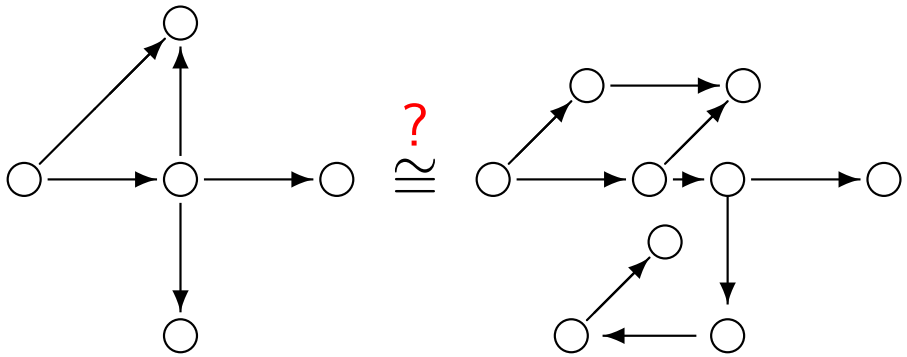


Integrated analysis

- Qualitative verification can now be complemented by quantitative verification:

Specification matching

With what probability
does system behaviour
match its specification?

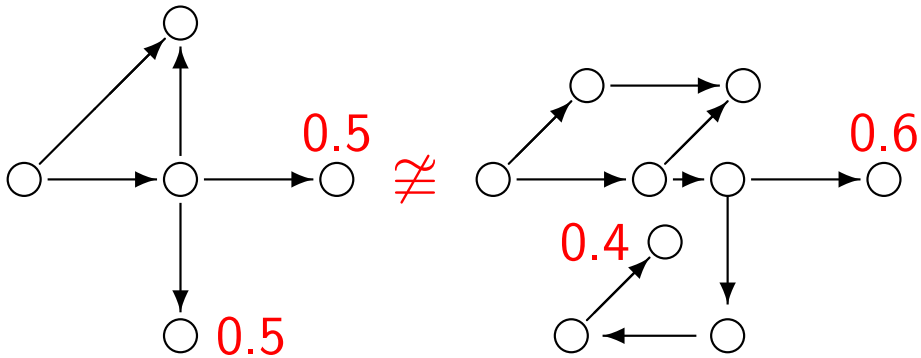


Integrated analysis

- Qualitative verification can now be complemented by quantitative verification:

Specification matching

Does the "*frequency profile*" of the system match that of the specification?

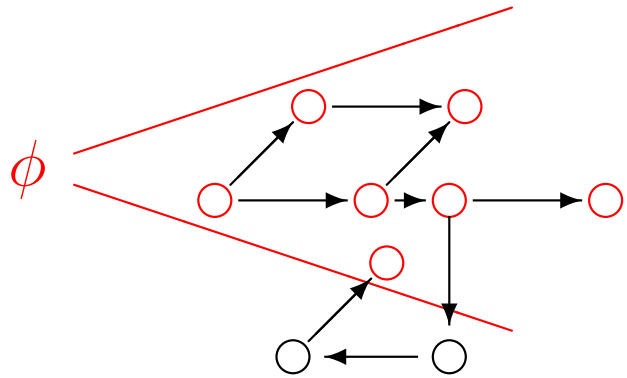


Integrated analysis

- Qualitative verification can now be complemented by quantitative verification:

Model checking

Does a given property ϕ hold within the system with a given probability?

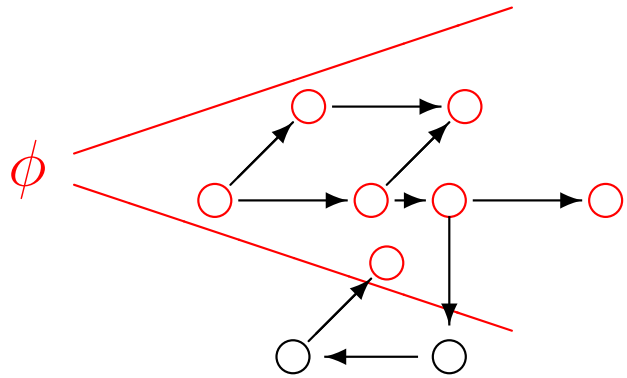


Integrated analysis

- Qualitative verification can now be complemented by quantitative verification:

Model checking

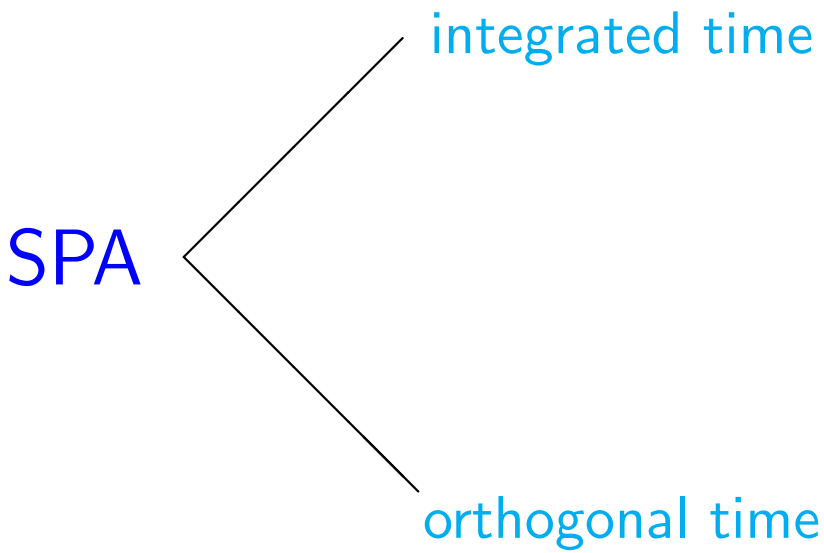
For a given starting state
how long is it until
a given property ϕ holds?



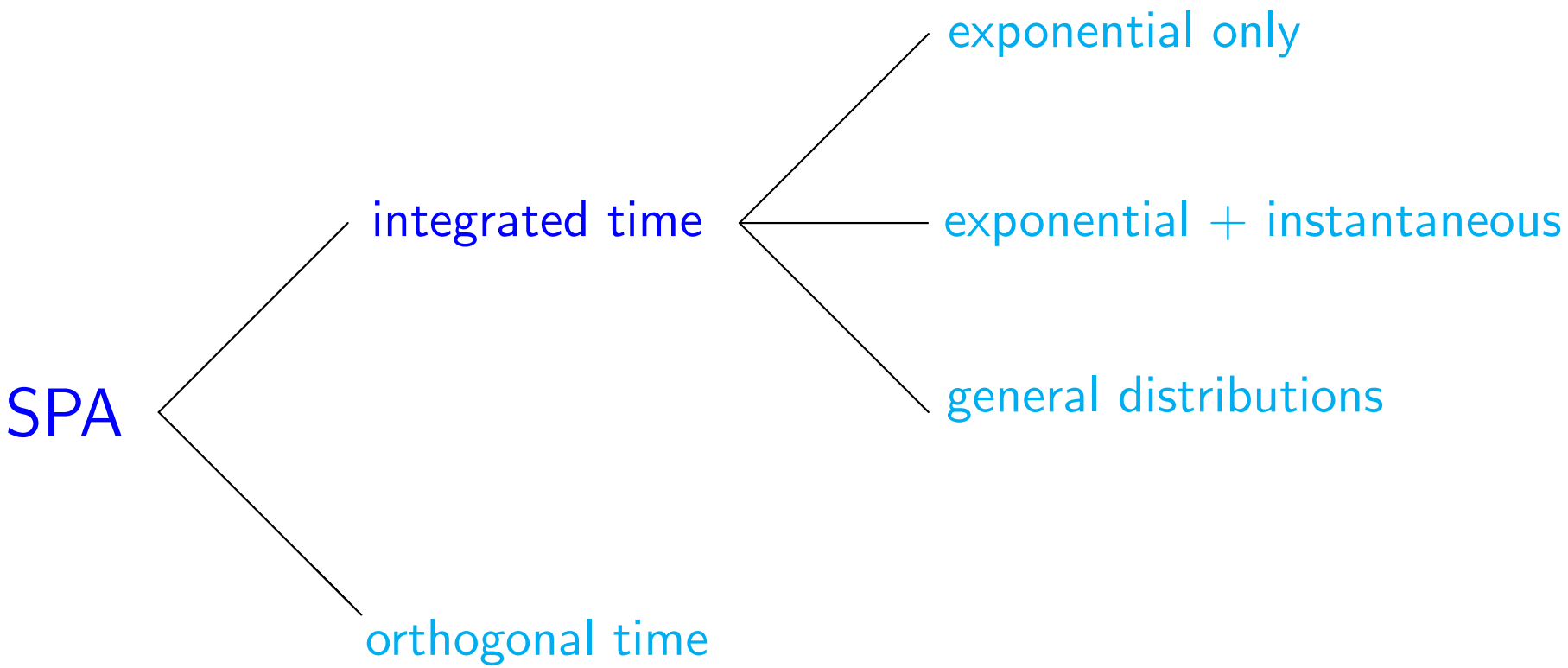
SPA Languages

SPA

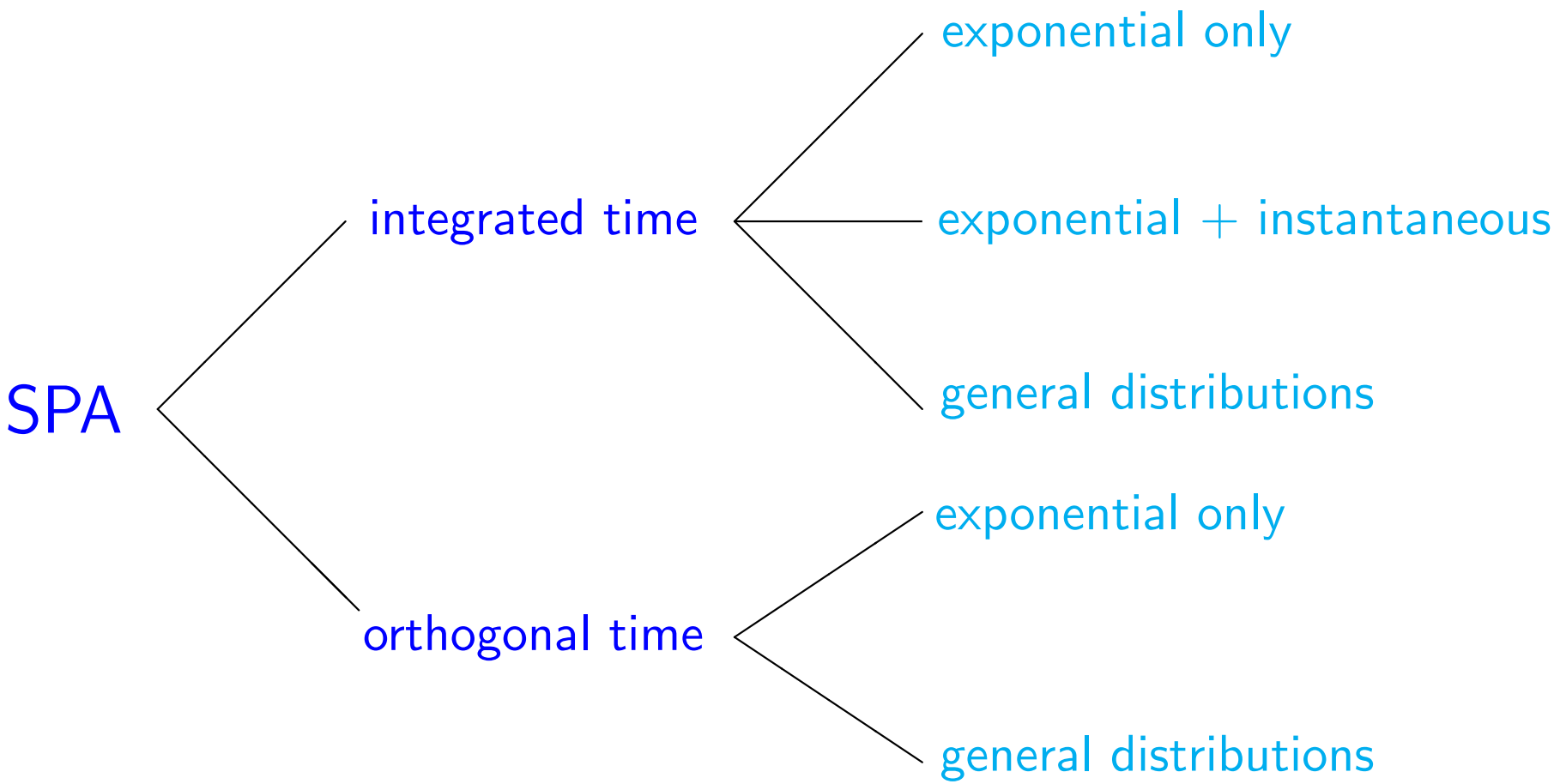
SPA Languages



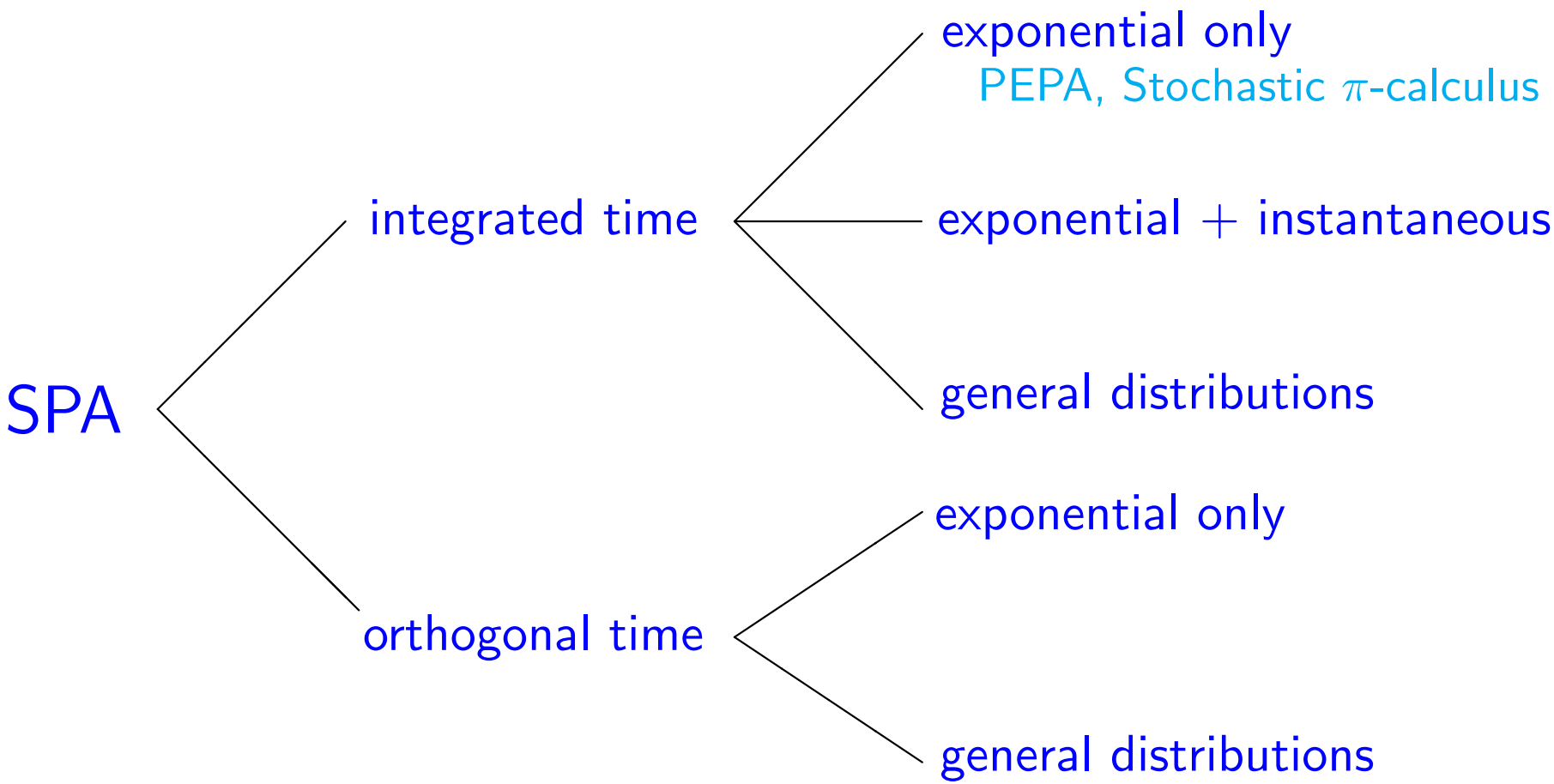
SPA Languages



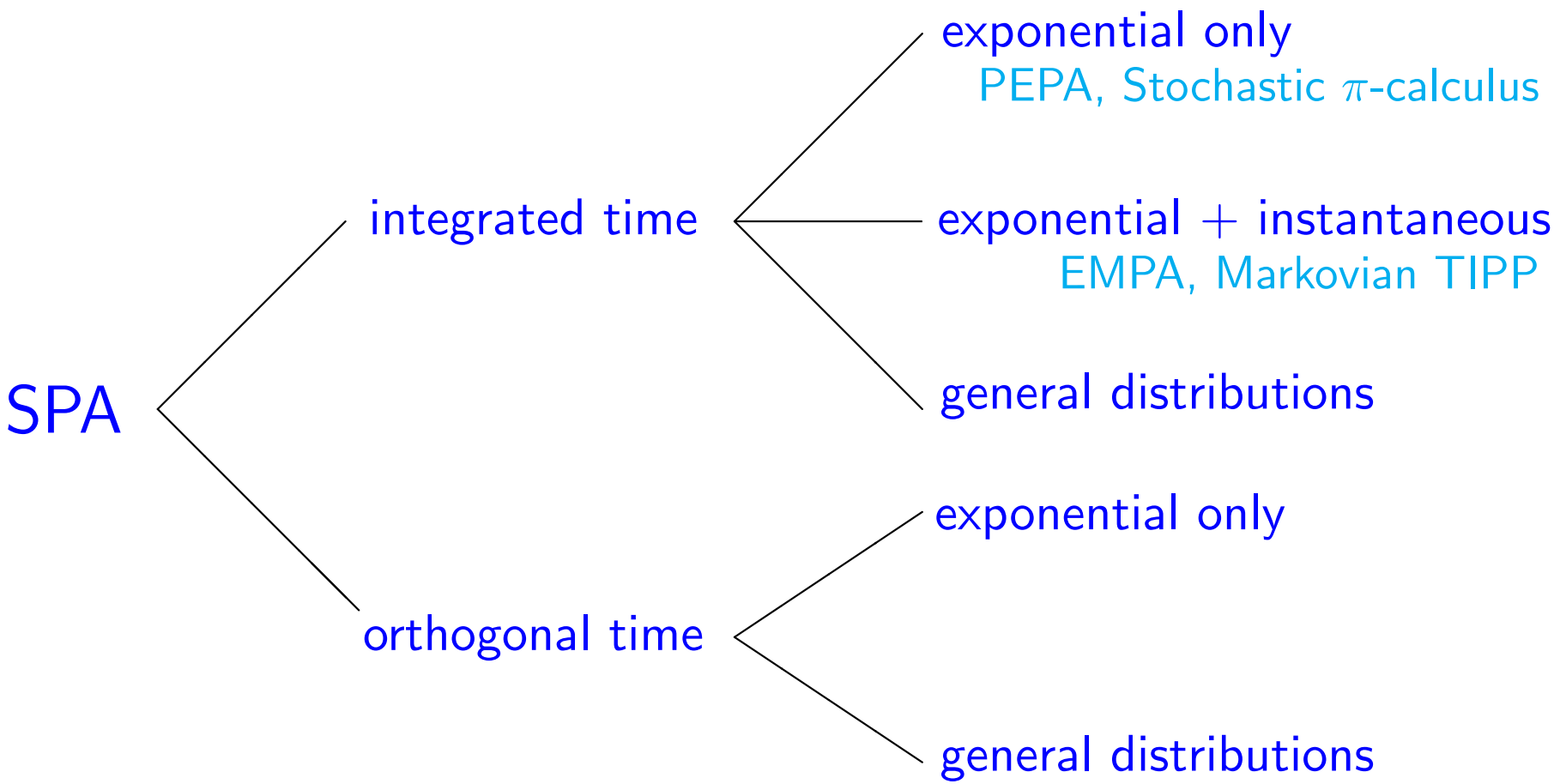
SPA Languages



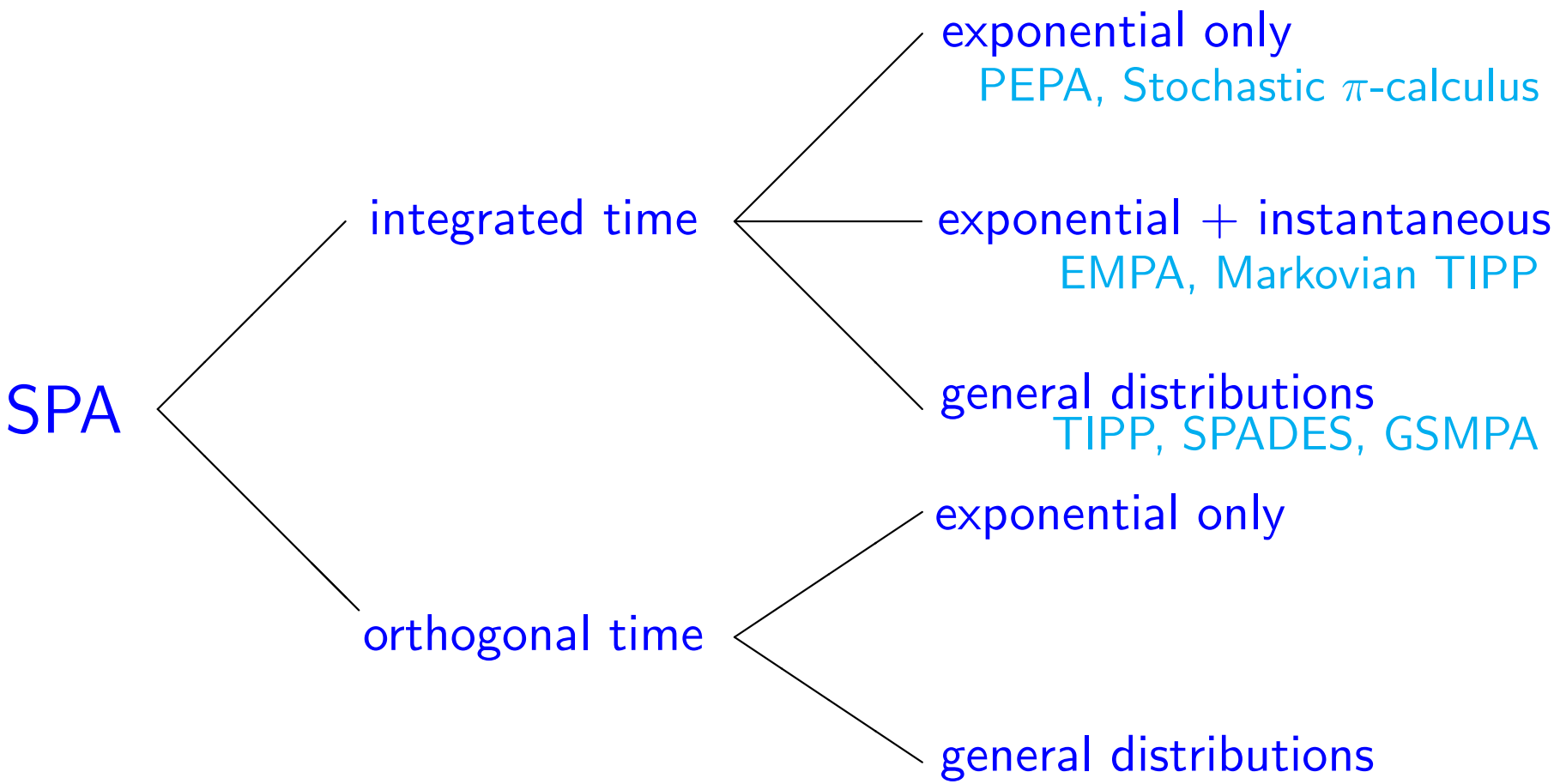
SPA Languages



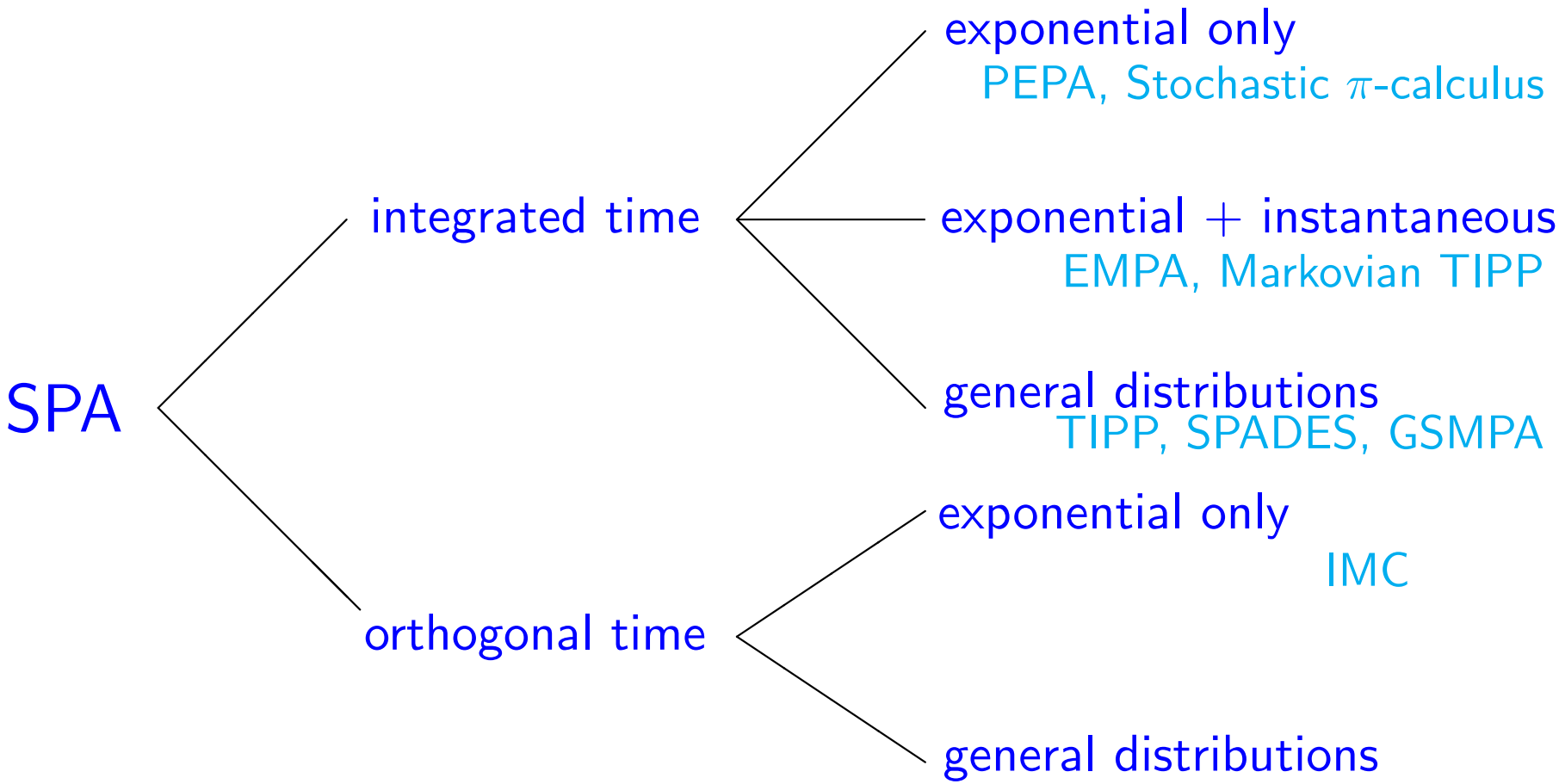
SPA Languages



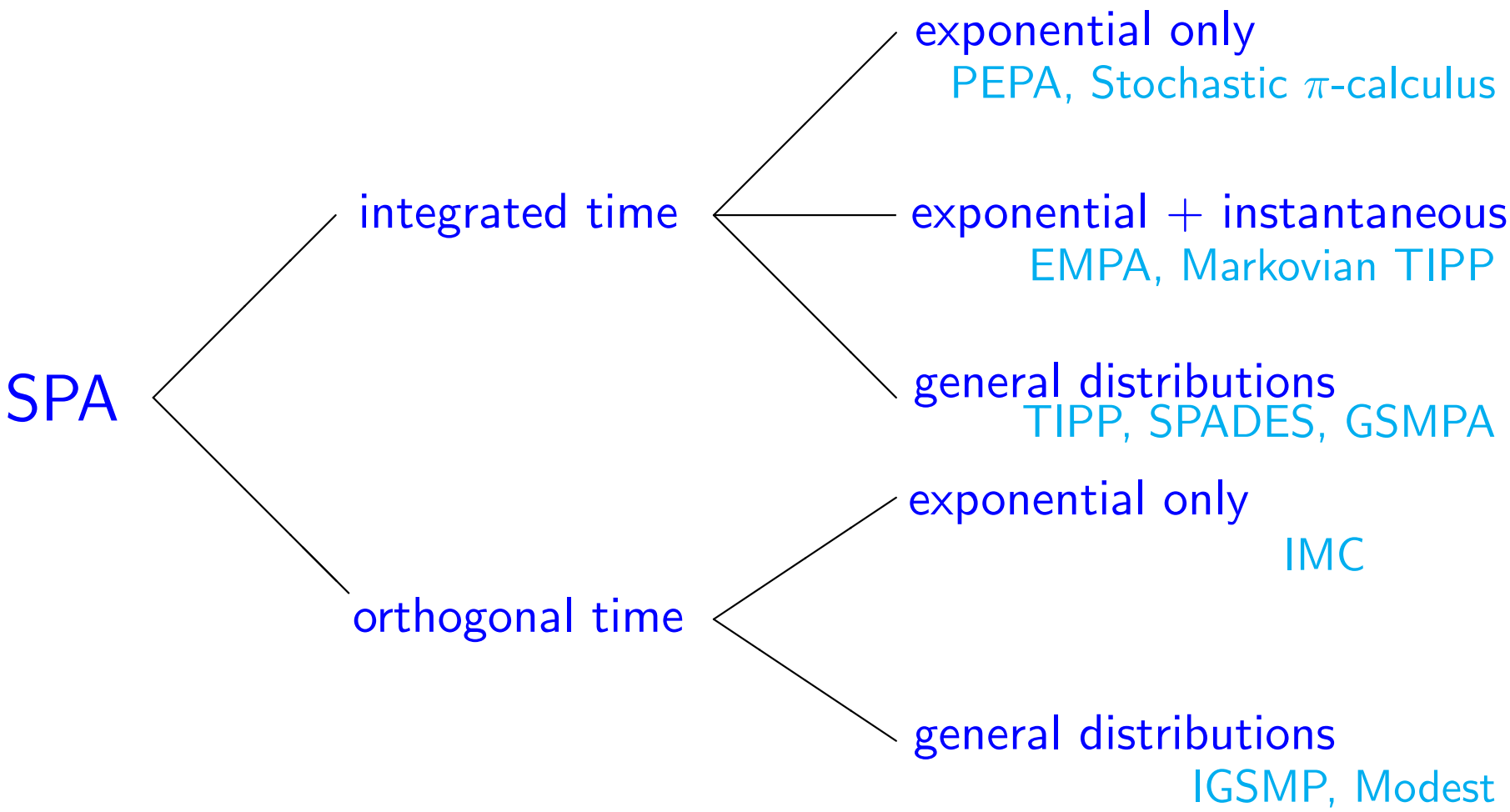
SPA Languages



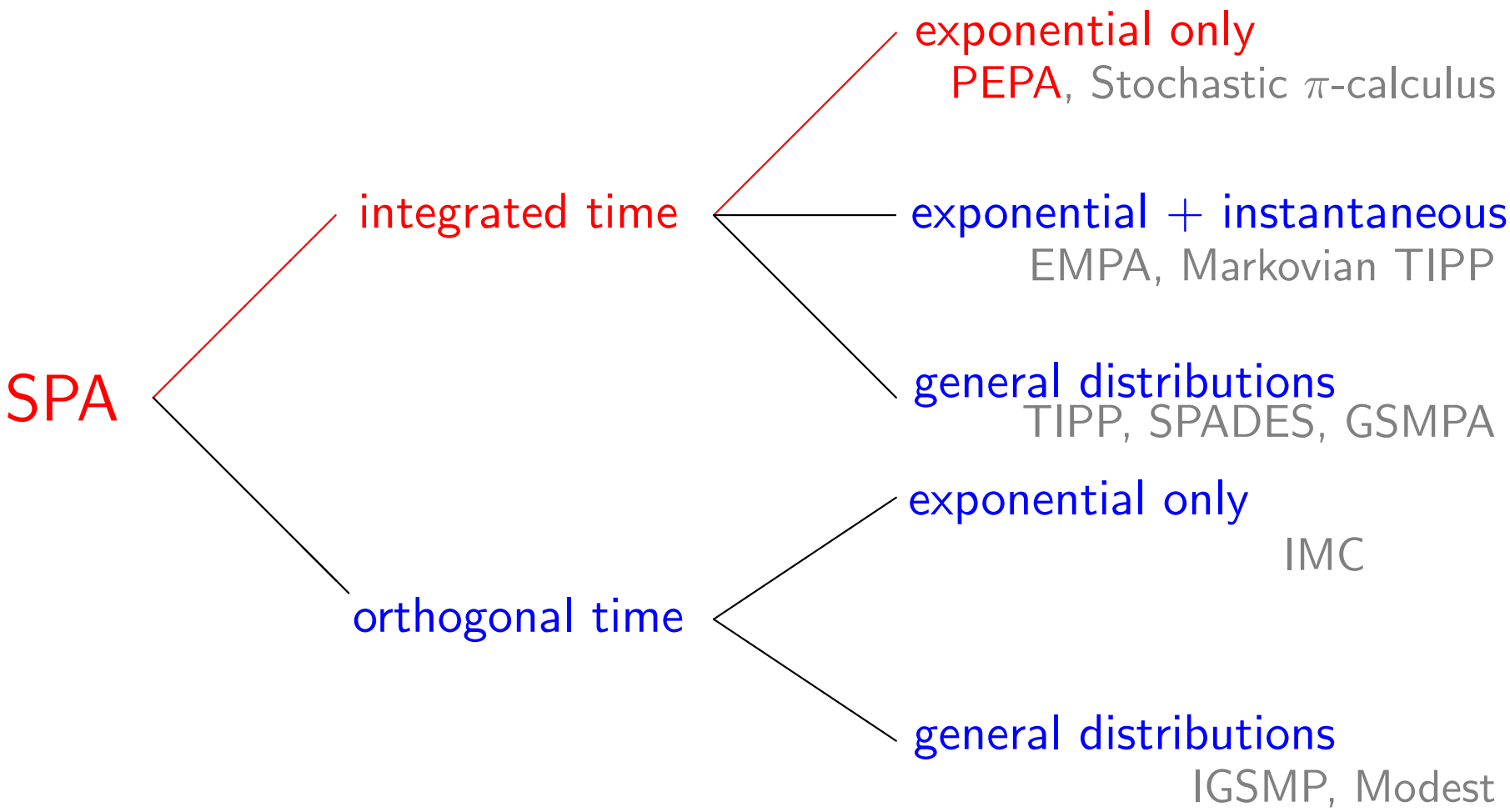
SPA Languages



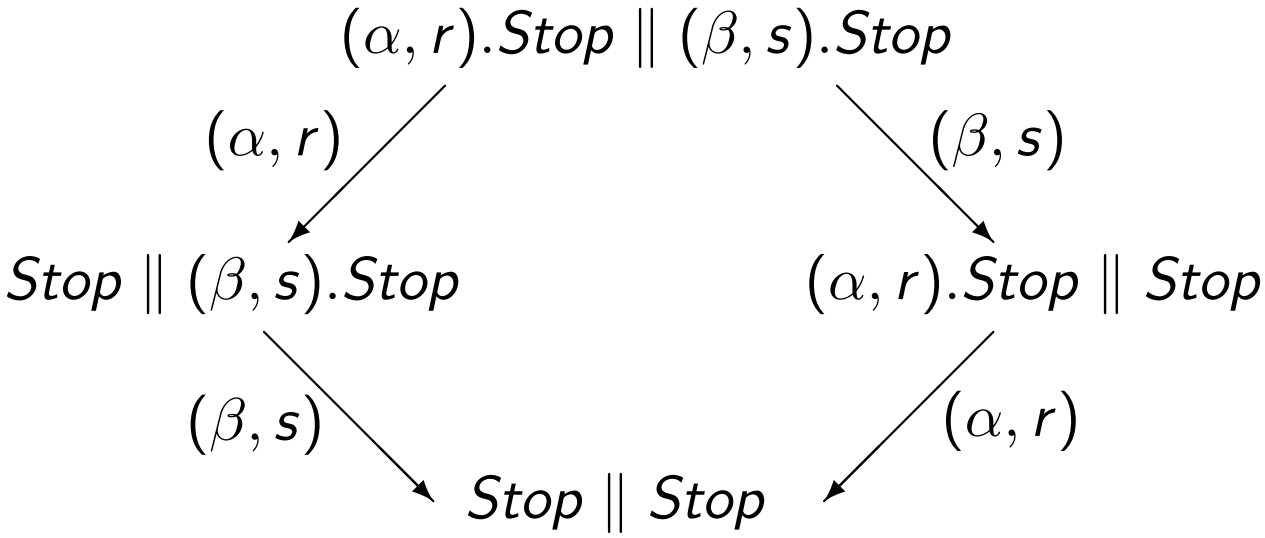
SPA Languages



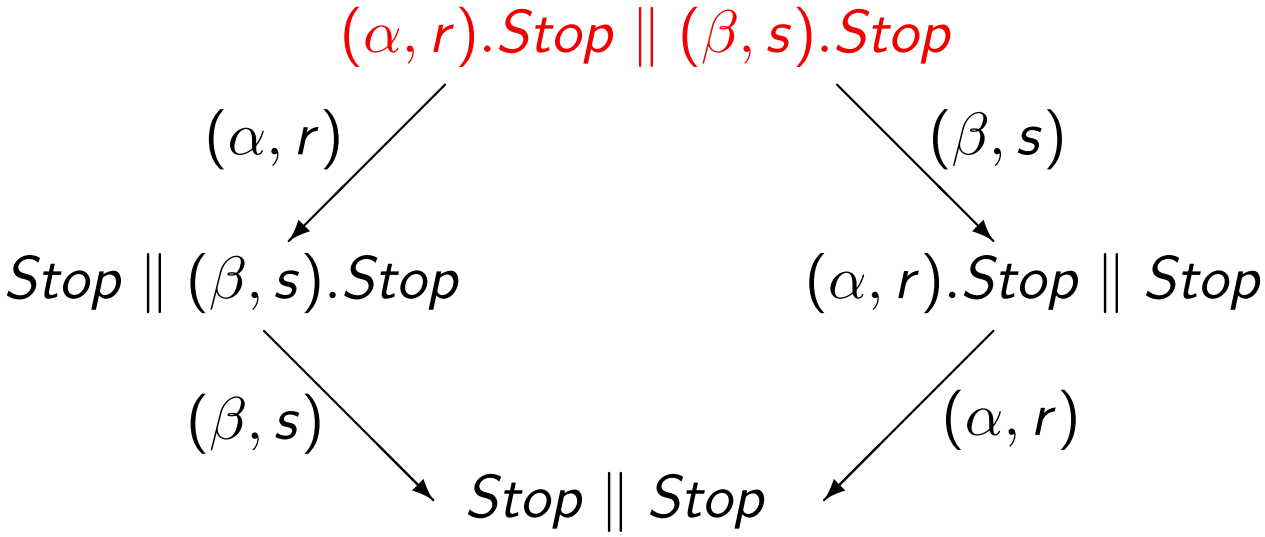
SPA Languages



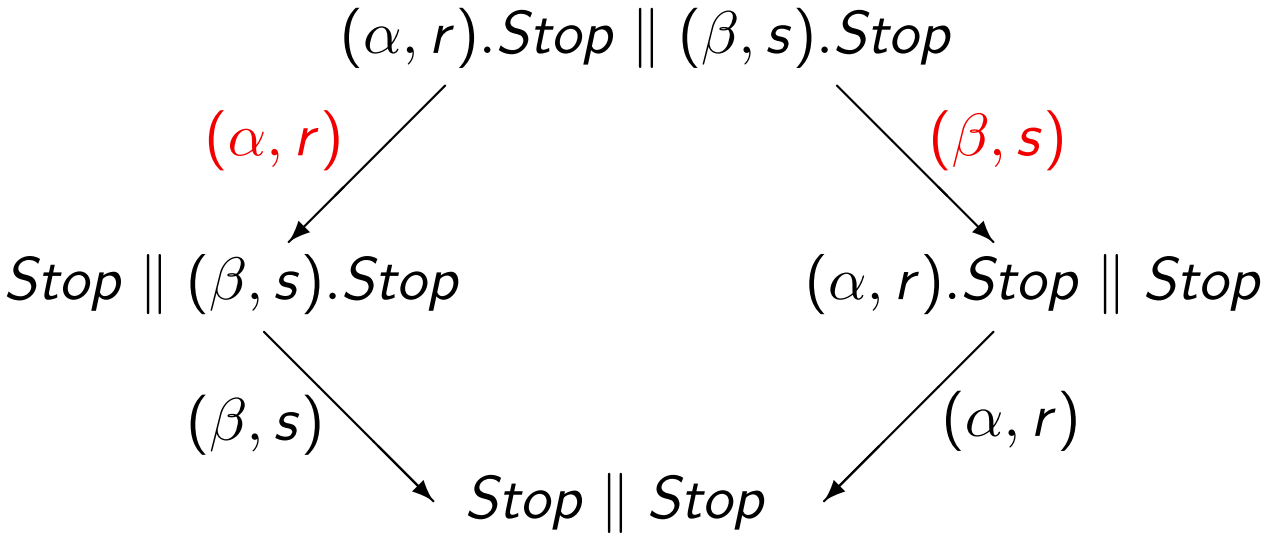
The Importance of Being Exponential



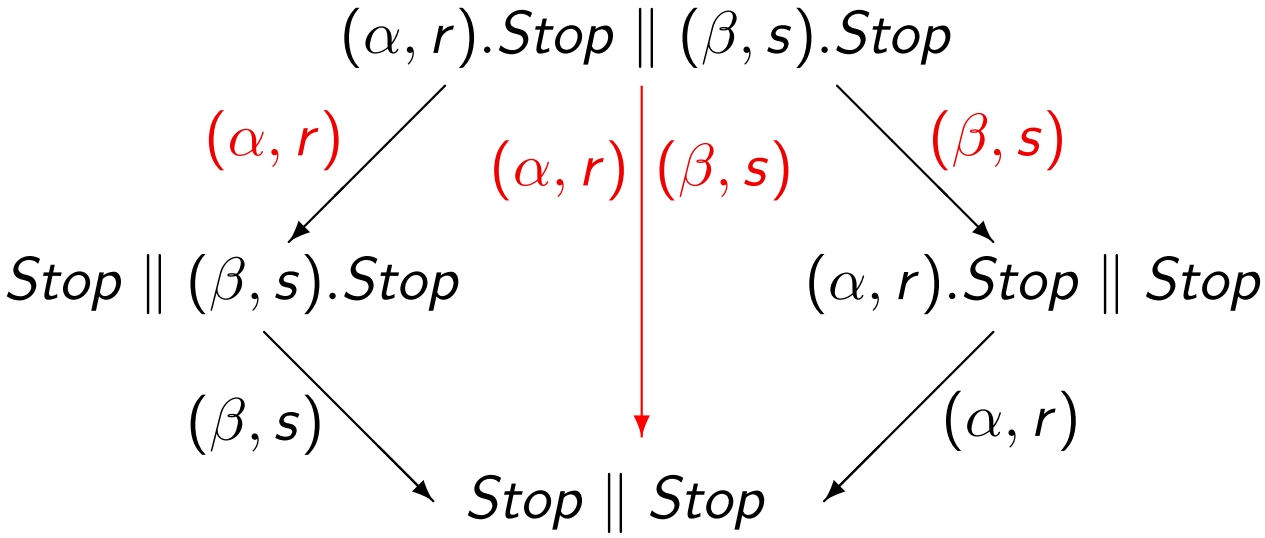
The Importance of Being Exponential



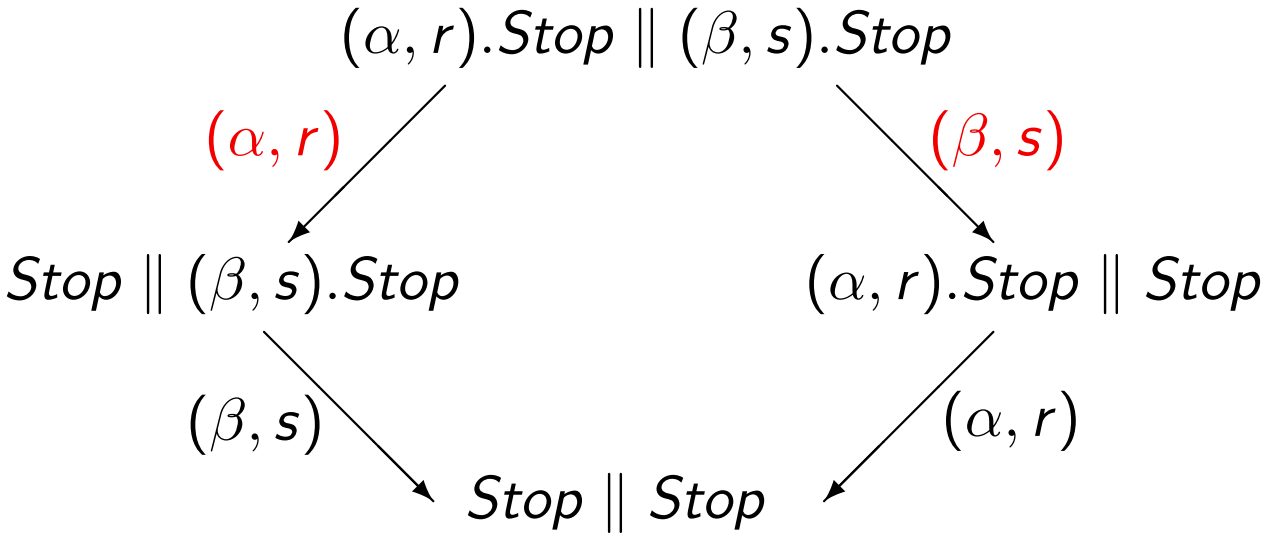
The Importance of Being Exponential



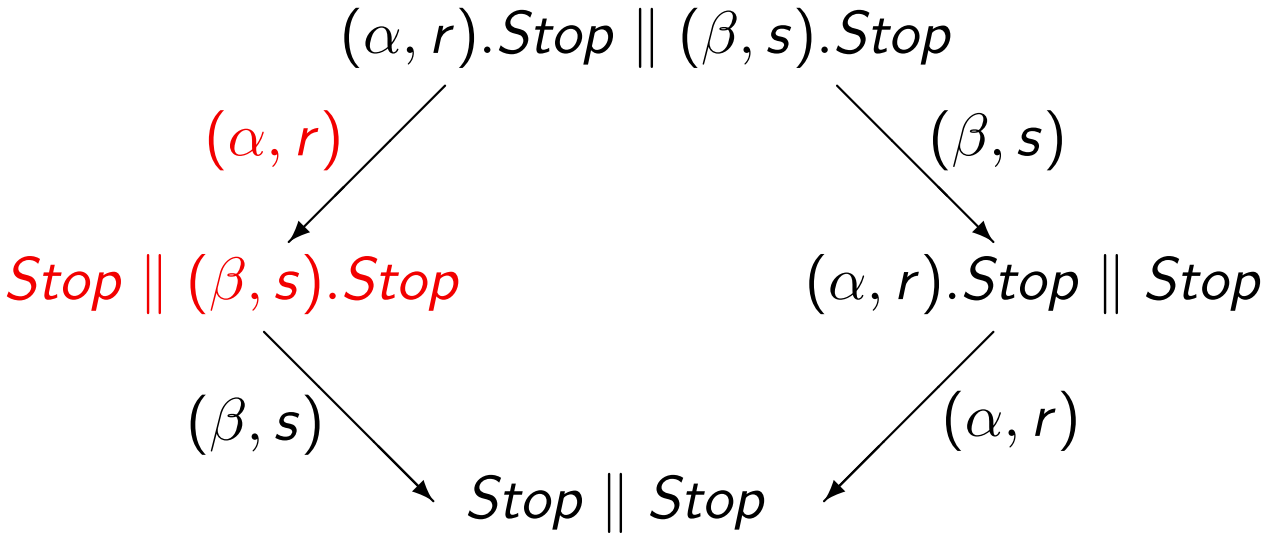
The Importance of Being Exponential



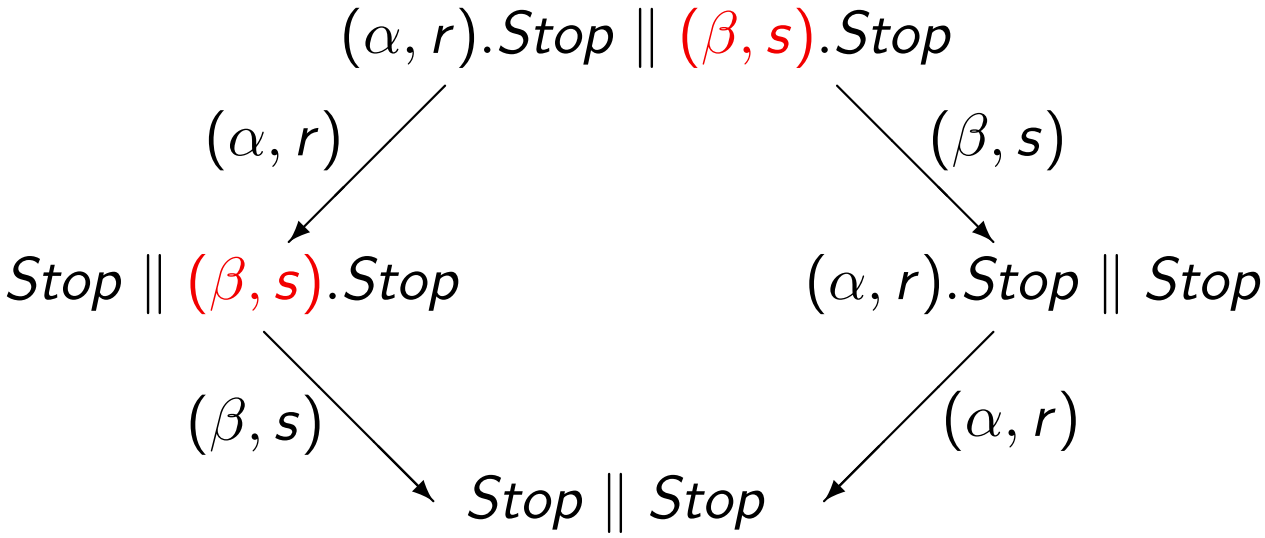
The Importance of Being Exponential



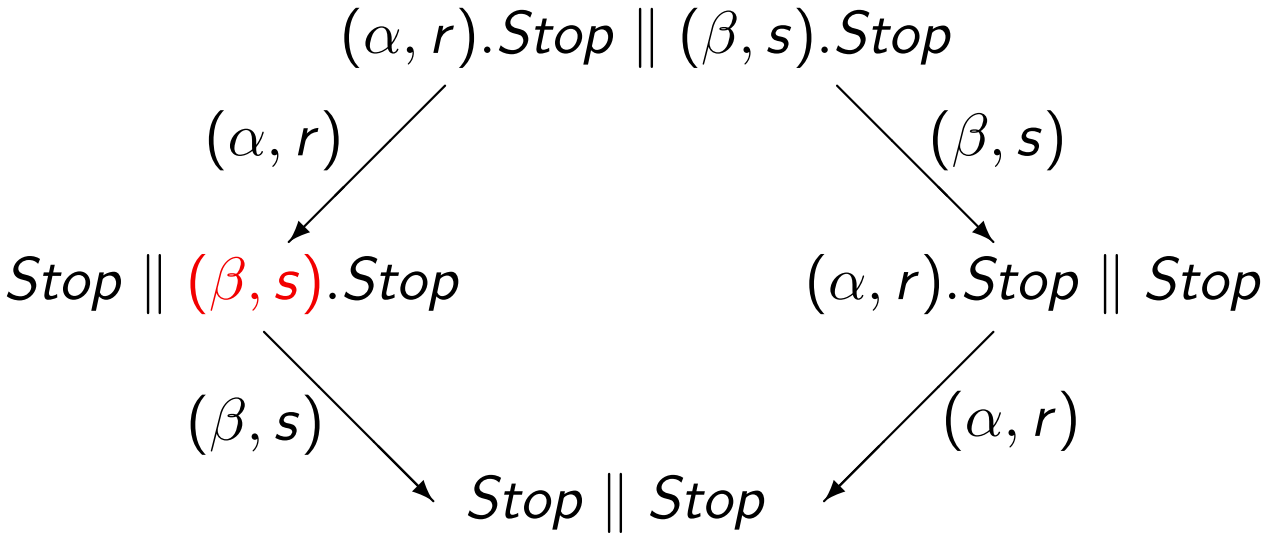
The Importance of Being Exponential



The Importance of Being Exponential

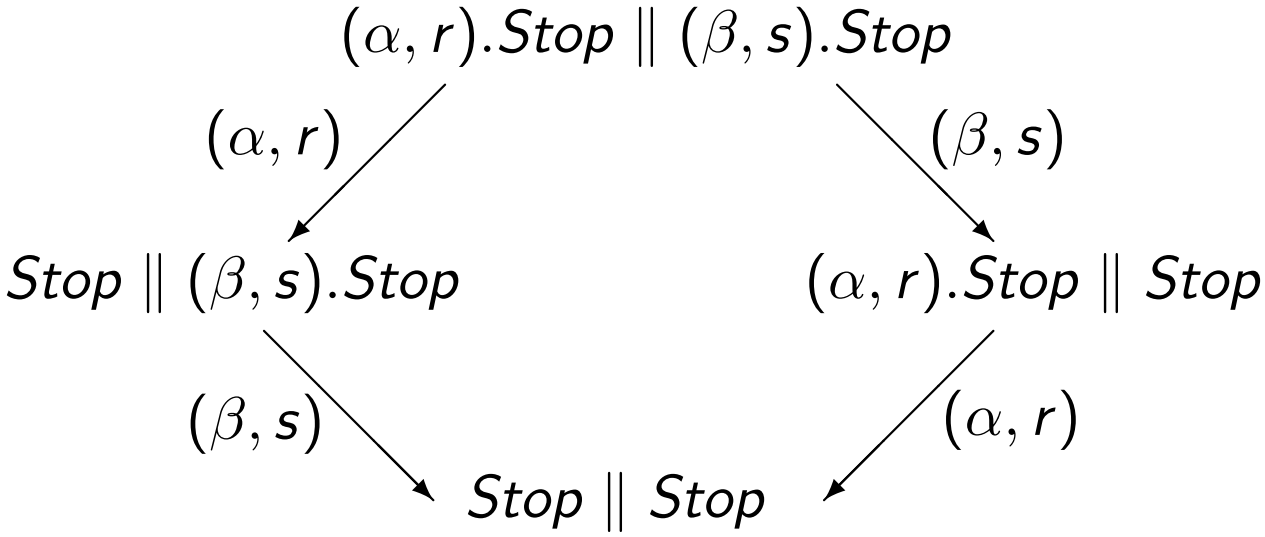


The Importance of Being Exponential



The memoryless property of the negative exponential distribution means that residual times do not need to be recorded.

The Importance of Being Exponential



We retain the expansion law of classical process algebra:

$$(\alpha, r).Stop \parallel (\beta, s).Stop =$$
$$(\alpha, r).(\beta, s).(Stop \parallel Stop) + (\beta, s).(\alpha, r).(Stop \parallel Stop)$$

only if the negative exponential distribution is assumed.

Parallel Composition

- Parellel composition is the basis of the compositionality in a process algebra

Parallel Composition

- Parellel composition is the basis of the compositionality in a process algebra — it defines which components interact and how.

Parallel Composition

- Parellel composition is the basis of the compositionality in a process algebra — it defines which components interact and how.
- In classical process algebra is it often associated with communication.

Parallel Composition

- Parellel composition is the basis of the compositionality in a process algebra — it defines which components interact and how.
- In classical process algebra is it often associated with communication.
- When the activities of the process algebra have a duration the definition of parallel composition becomes more complex.

Who Synchronises...?

Even within classical process algebras there is variation in the interpretation of parallel composition:

Who Synchronises...?

Even within classical process algebras there is variation in the interpretation of parallel composition:

CCS-style

- Actions are partitioned into input and output pairs.
- Communication or synchronisation takes places between conjugate pairs.
- The resulting action has silent type τ.

Who Synchronises...?

Even within classical process algebras there is variation in the interpretation of parallel composition:

CCS-style

- Actions are partitioned into input and output pairs.
- Communication or synchronisation takes places between conjugate pairs.
- The resulting action has silent type τ.

Who Synchronises...?

Even within classical process algebras there is variation in the interpretation of parallel composition:

CCS-style

- Actions are partitioned into input and output pairs.
- Communication or synchronisation takes places between conjugate pairs.
- The resulting action has silent type τ.

Who Synchronises...?

Even within classical process algebras there is variation in the interpretation of parallel composition:

CCS-style

- Actions are partitioned into input and output pairs.
- Communication or synchronisation takes places between conjugate pairs.
- The resulting action has silent type τ.

CSP-style

- No distinction between input and output actions.
- Communication or synchronisation takes place on the basis of shared names.
- The resulting action has the same name.

Who Synchronises...?

Even within classical process algebras there is variation in the interpretation of parallel composition:

CCS-style

- Actions are partitioned into input and output pairs.
- Communication or synchronisation takes places between conjugate pairs.
- The resulting action has silent type τ.

CSP-style

- No distinction between input and output actions.
- Communication or synchronisation takes place on the basis of shared names.
- The resulting action has the same name.

Who Synchronises...?

Even within classical process algebras there is variation in the interpretation of parallel composition:

CCS-style

- Actions are partitioned into input and output pairs.
- Communication or synchronisation takes places between conjugate pairs.
- The resulting action has silent type τ.

CSP-style

- No distinction between input and output actions.
- Communication or synchronisation takes place on the basis of shared names.
- The resulting action has the same name.

Who Synchronises...?

Even within classical process algebras there is variation in the interpretation of parallel composition:

CCS-style

- Actions are partitioned into input and output pairs.
- Communication or synchronisation takes places between conjugate pairs.
- The resulting action has silent type τ.

CSP-style

- No distinction between input and output actions.
- Communication or synchronisation takes place on the basis of shared names.
- The resulting action has the same name.

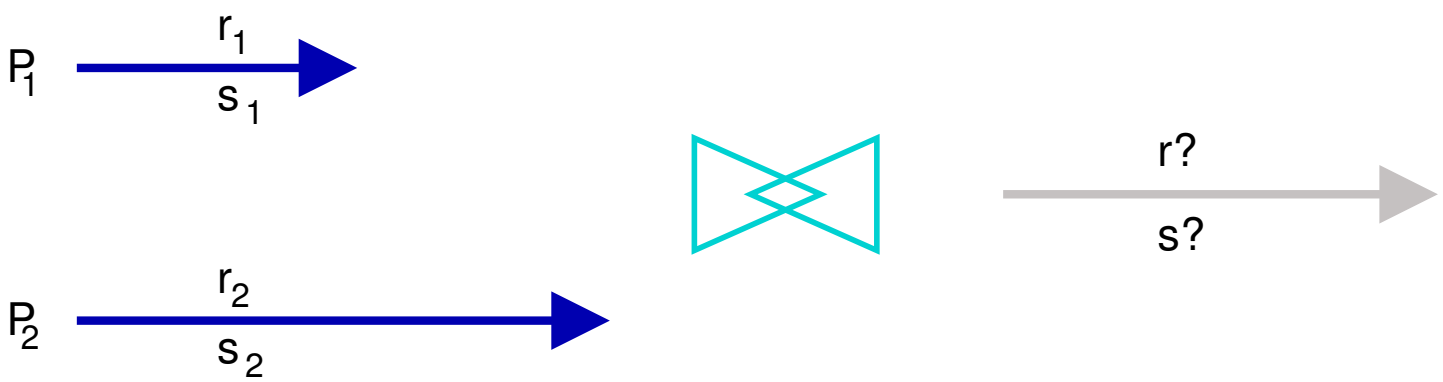
Most stochastic process algebras adopt CSP-style synchronisation.

Timed Synchronisation

- The issue of what it means for two timed activities to synchronise is a vexed one....

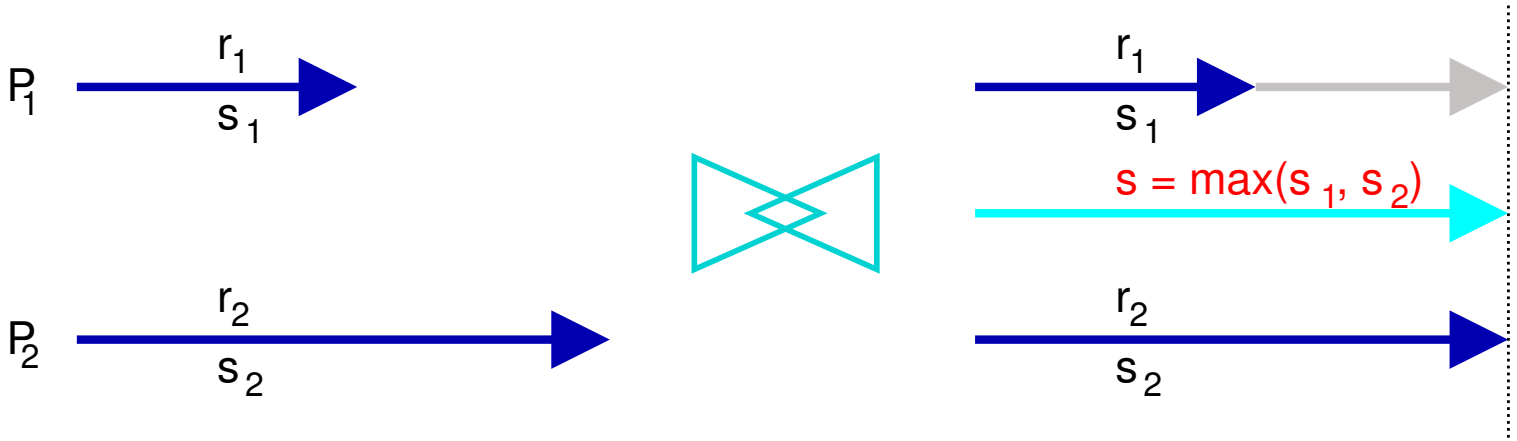
Timed Synchronisation

- The issue of what it means for two timed activities to synchronise is a vexed one....



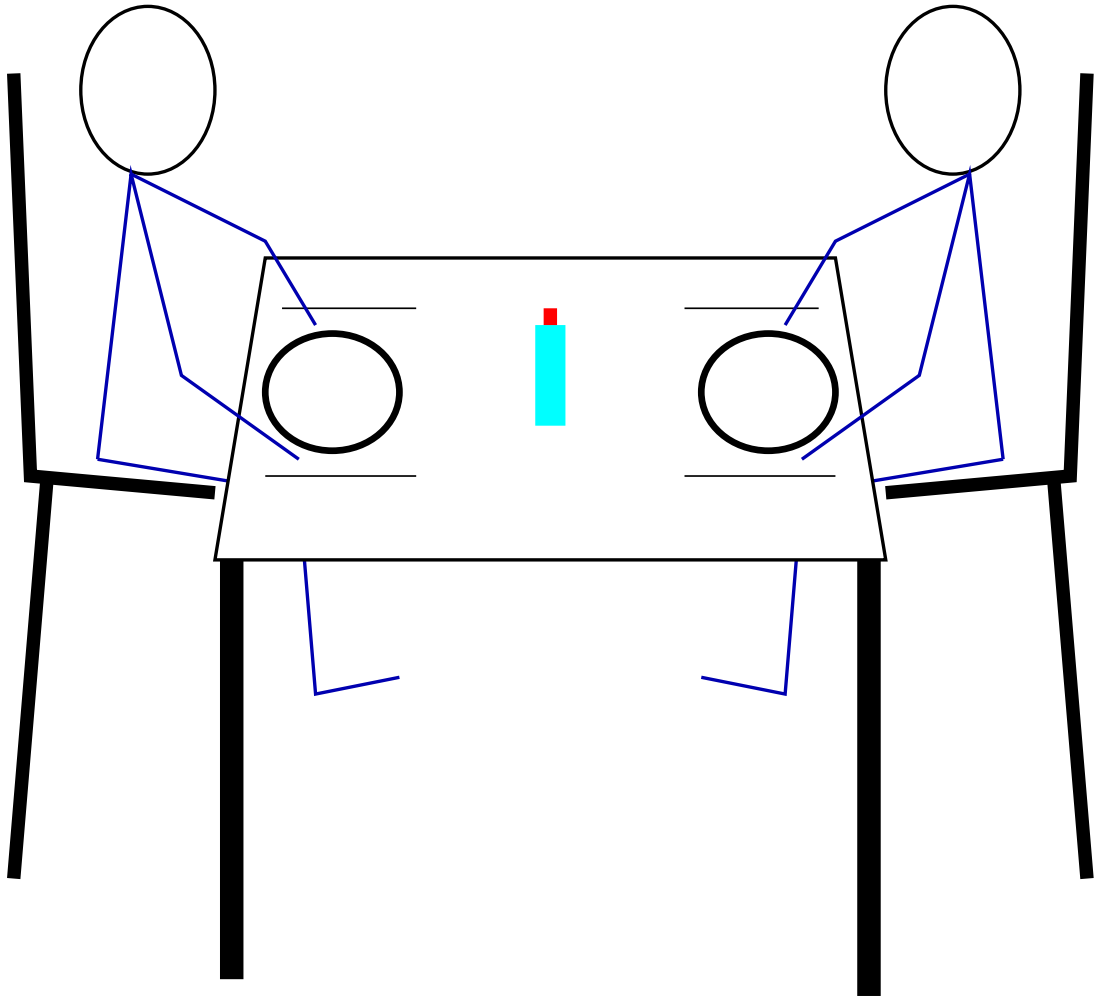
Timed Synchronisation

- The issue of what it means for two timed activities to synchronise is a vexed one....



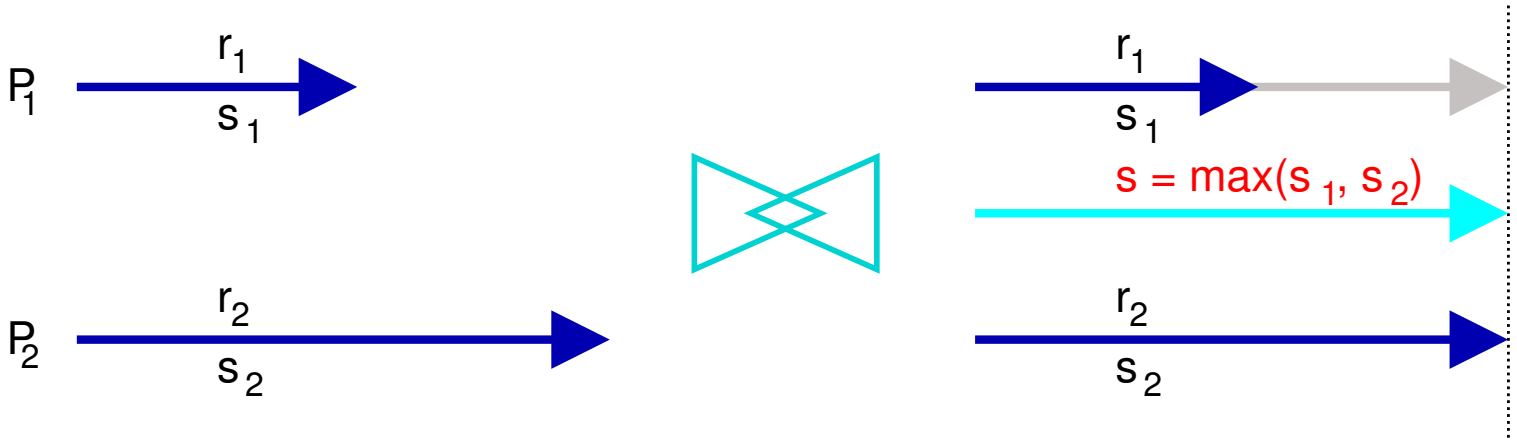
Barrier Synchronisation

Timed Synchronisation



Timed Synchronisation

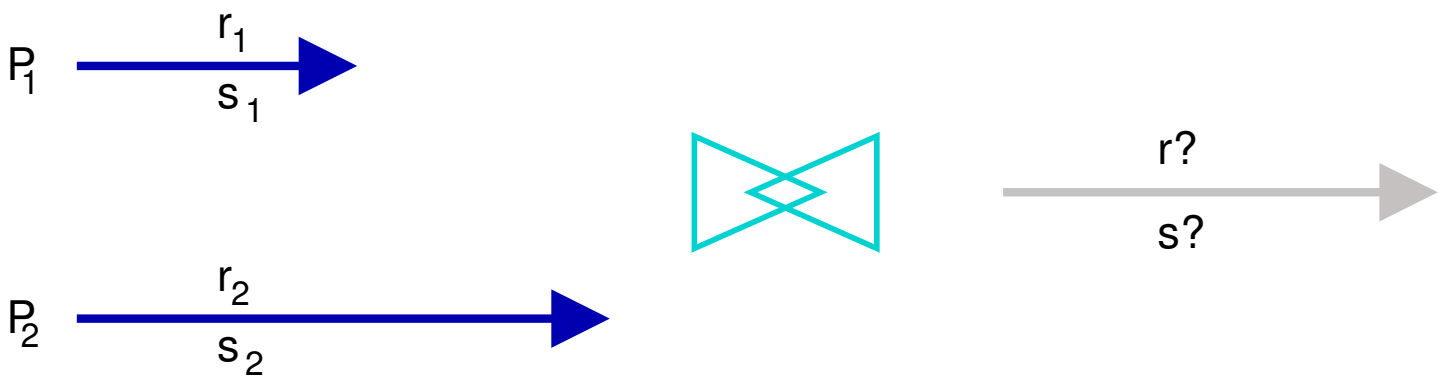
- The issue of what it means for two timed activities to synchronise is a vexed one....



s is no longer exponentially distributed

Timed Synchronisation

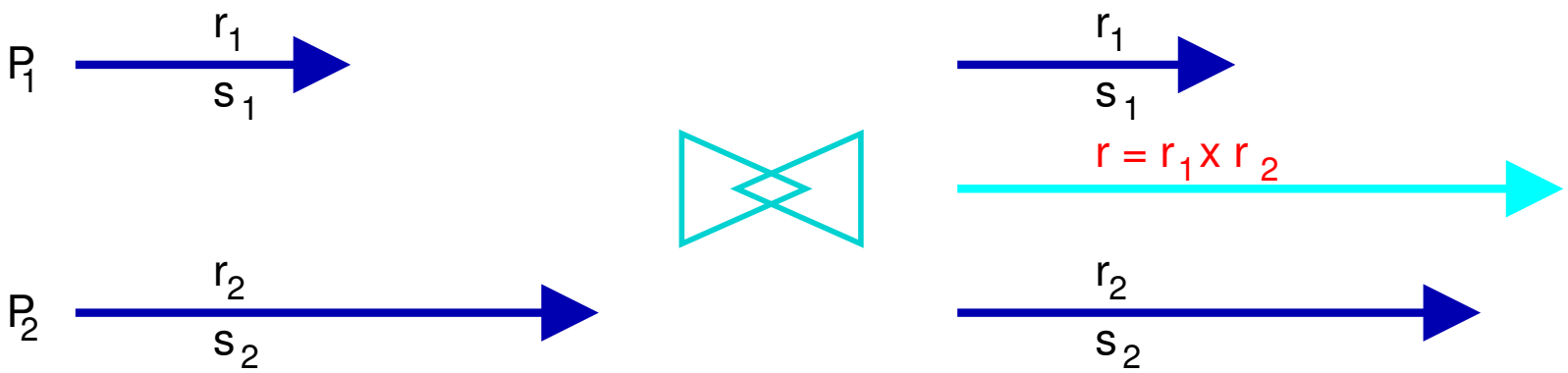
- The issue of what it means for two timed activities to synchronise is a vexed one....



algebraic considerations limit choices

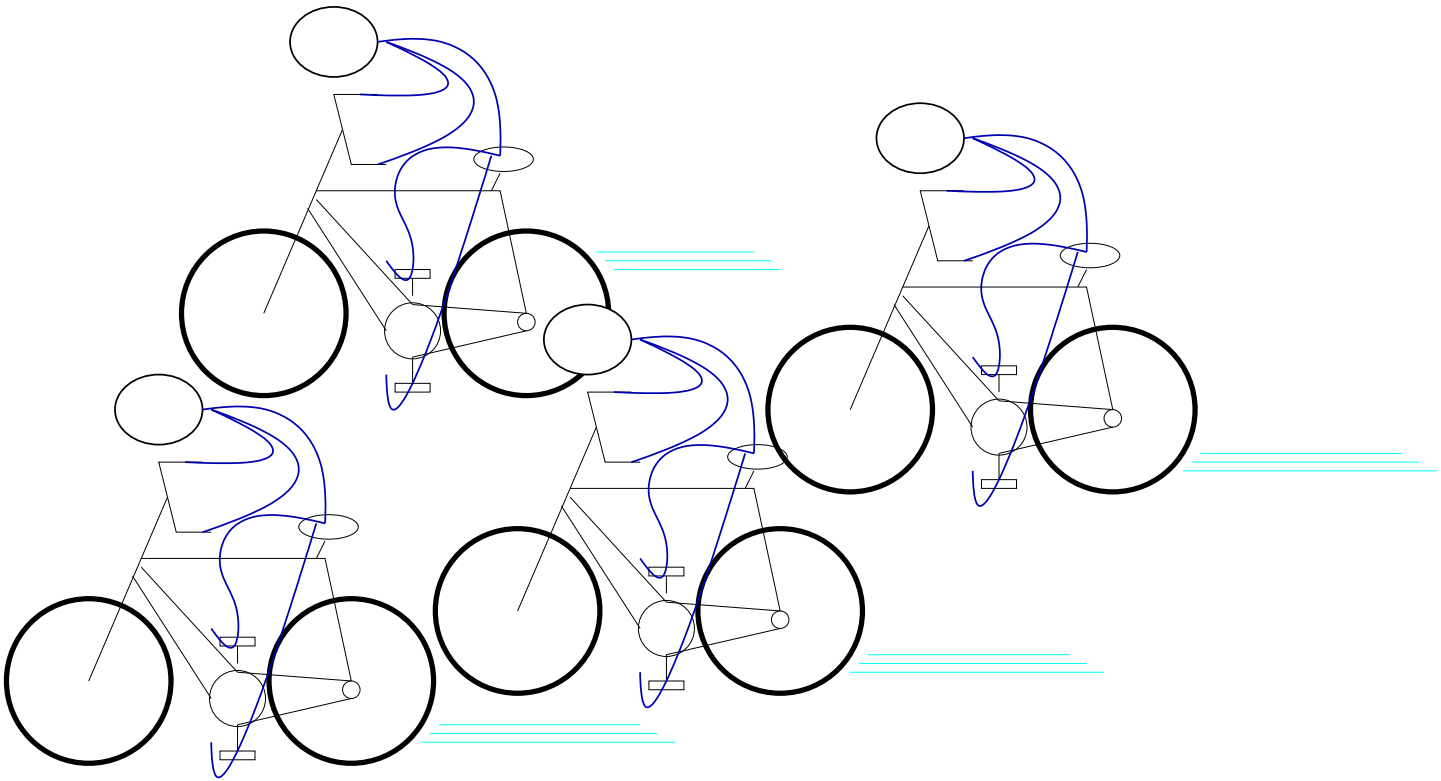
Timed Synchronisation

- The issue of what it means for two timed activities to synchronise is a vexed one....



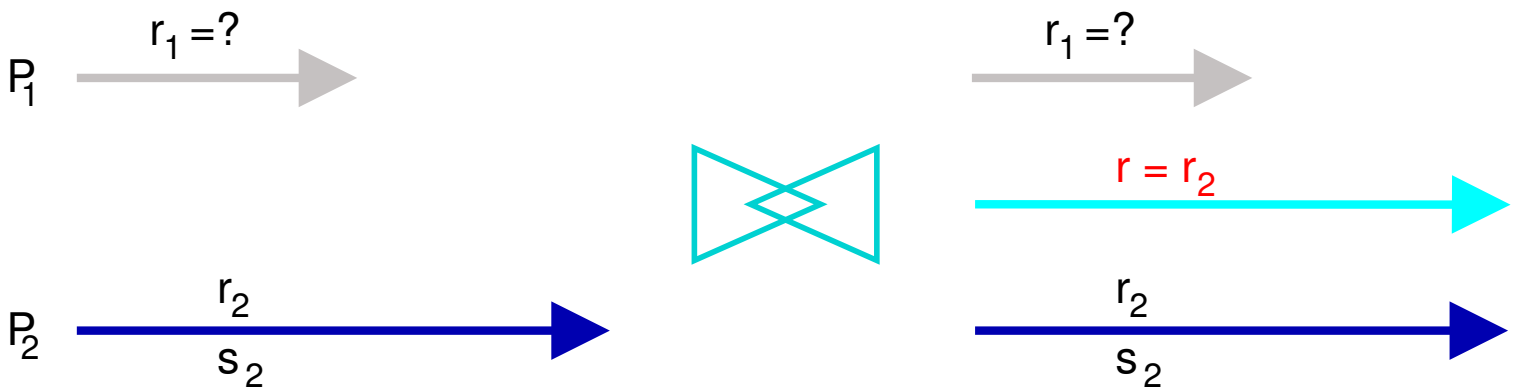
TIPP: new rate is product of individual rates

Timed Synchronisation



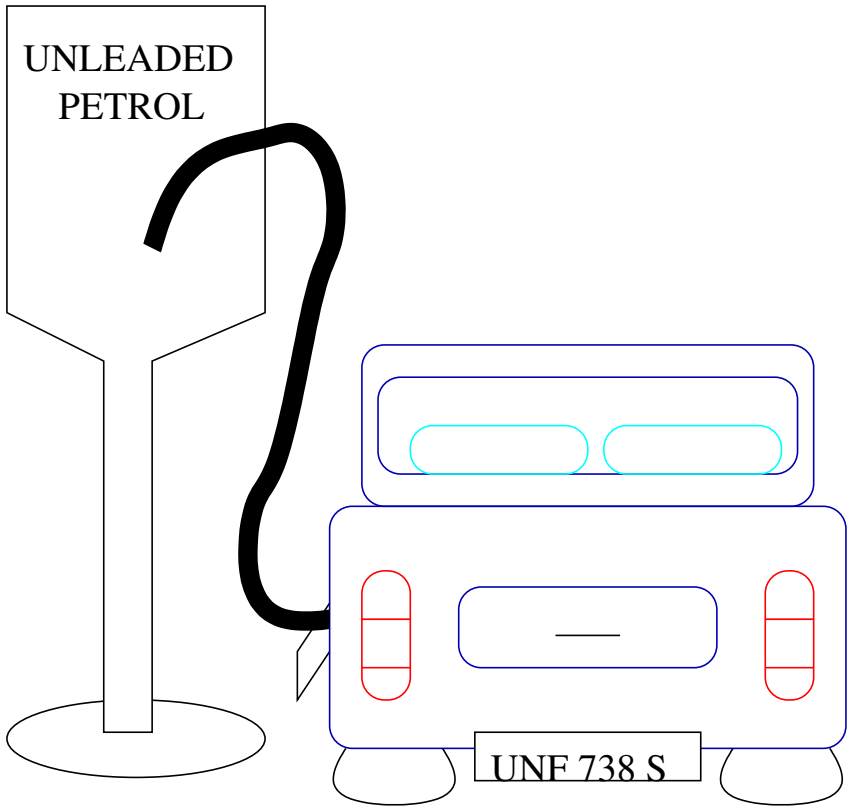
Timed Synchronisation

- The issue of what it means for two timed activities to synchronise is a vexed one....



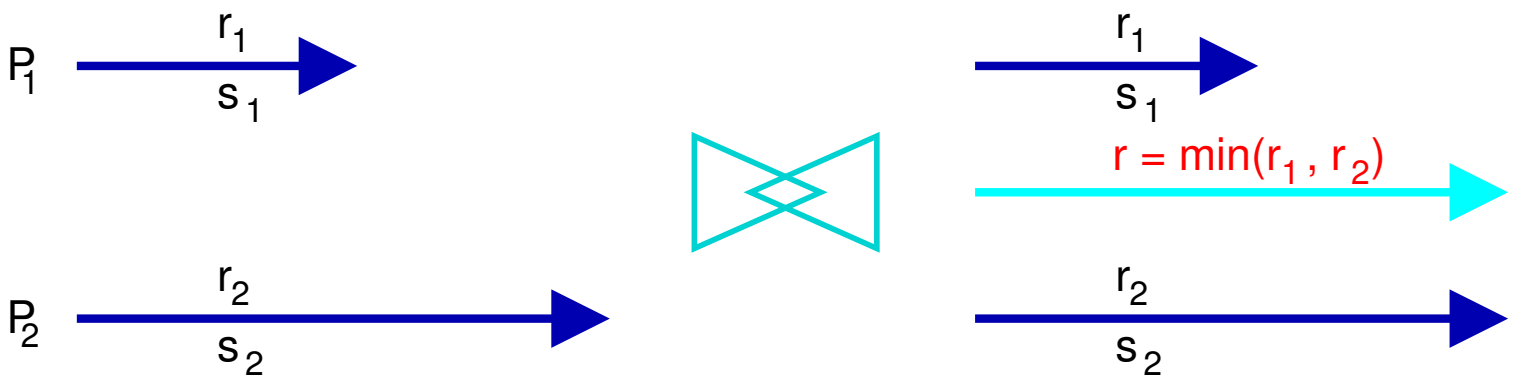
EMPA: one participant is passive

Timed Synchronisation



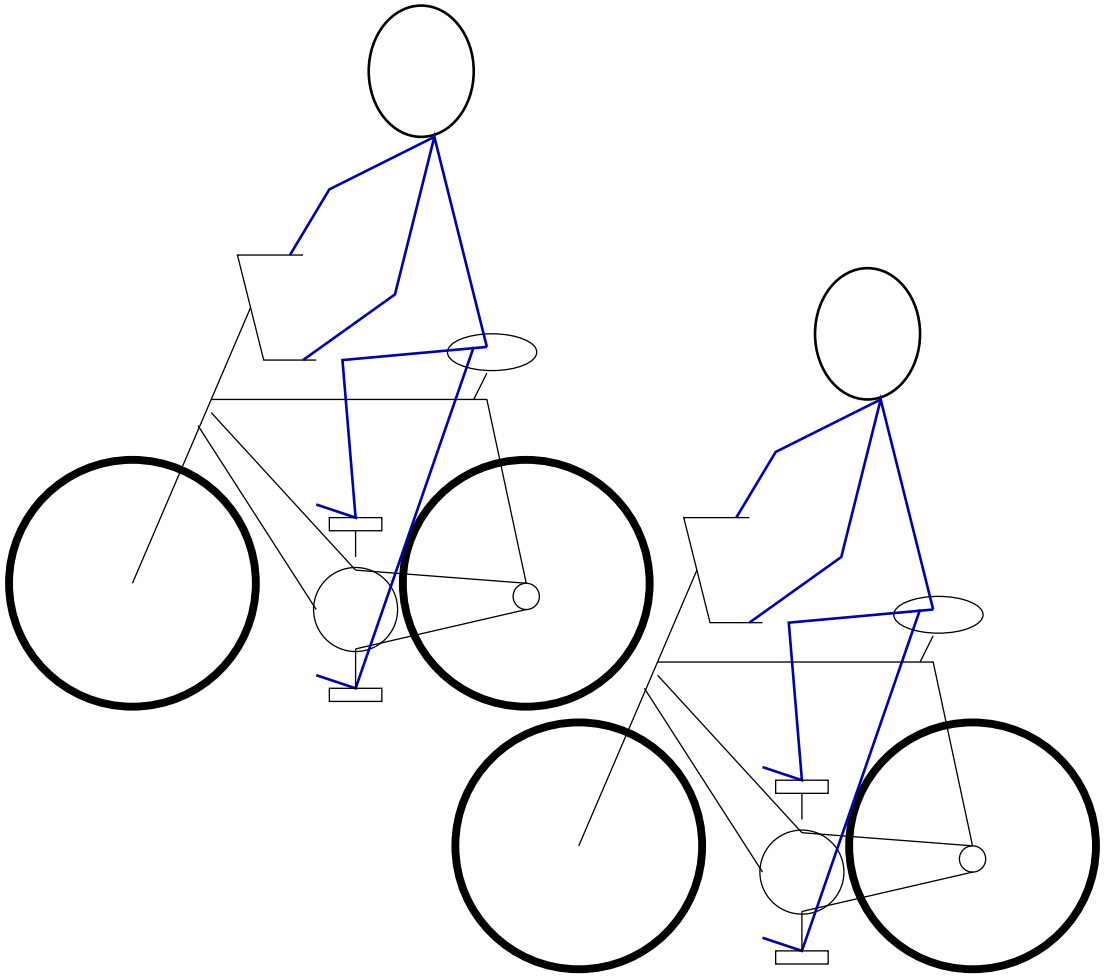
Timed Synchronisation

- The issue of what it means for two timed activities to synchronise is a vexed one....



bounded capacity: new rate is the minimum of the rates

Timed Synchronisation



Cooperation in PEPA

- In PEPA each component has a bounded capacity to carry out activities of any particular type, determined by the apparent rate for that type.

Cooperation in PEPA

- In PEPA each component has a bounded capacity to carry out activities of any particular type, determined by the apparent rate for that type.
- Synchronisation, or cooperation cannot make a component exceed its bounded capacity.

Cooperation in PEPA

- In PEPA each component has a bounded capacity to carry out activities of any particular type, determined by the apparent rate for that type.
- Synchronisation, or cooperation cannot make a component exceed its bounded capacity.
- Thus the apparent rate of a cooperation is the minimum of the apparent rates of the co-operands.

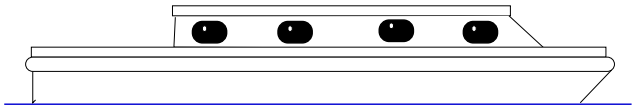
Outline

PEPA Case Studies (1)

- Multiprocessor access-contention protocols (Gilmore, Hillston and Ribaudo, Edinburgh and Turin)
- Protocols for fault-tolerant systems (Clark, Gilmore, Hillston and Ribaudo, Edinburgh and Turin)
- Multimedia traffic characteristics (Bowman et al, Kent)
- Database systems (The STEADY group, Heriot-Watt University)
- Software Architectures (Pooley, Bradley and Thomas, Heriot-Watt and Durham)
- Switch behaviour in active networks (Hillston, Kloul and Mokhtari, Edinburgh and Versailles)

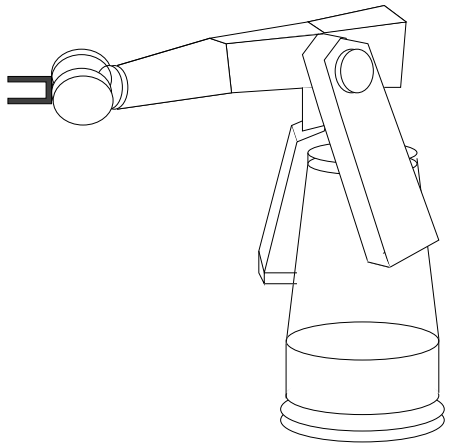
PEPA Case Studies (2)

- Locks and movable bridges in inland shipping in Belgium (Knapen, Hasselt)



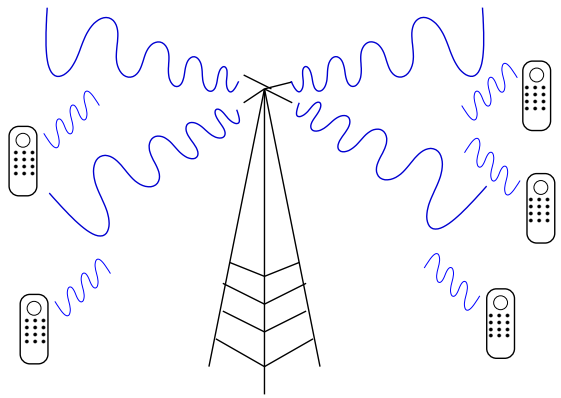
PEPA Case Studies (2)

- Locks and movable bridges in inland shipping in Belgium (Knapen, Hasselt)
- Robotic workcells (Holton, Gilmore and Hillston, Bradford and Edinburgh)



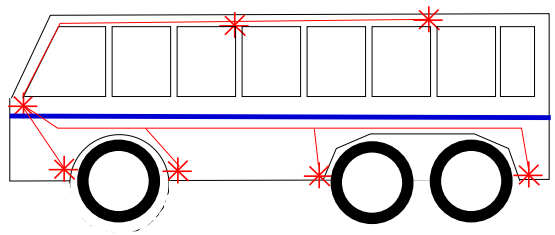
PEPA Case Studies (2)

- Locks and movable bridges in inland shipping in Belgium (Knapen, Hasselt)
- Robotic workcells (Holton, Gilmore and Hillston, Bradford and Edinburgh)
- Cellular telephone networks (Kloul, Fourneau and Valois, Versailles)

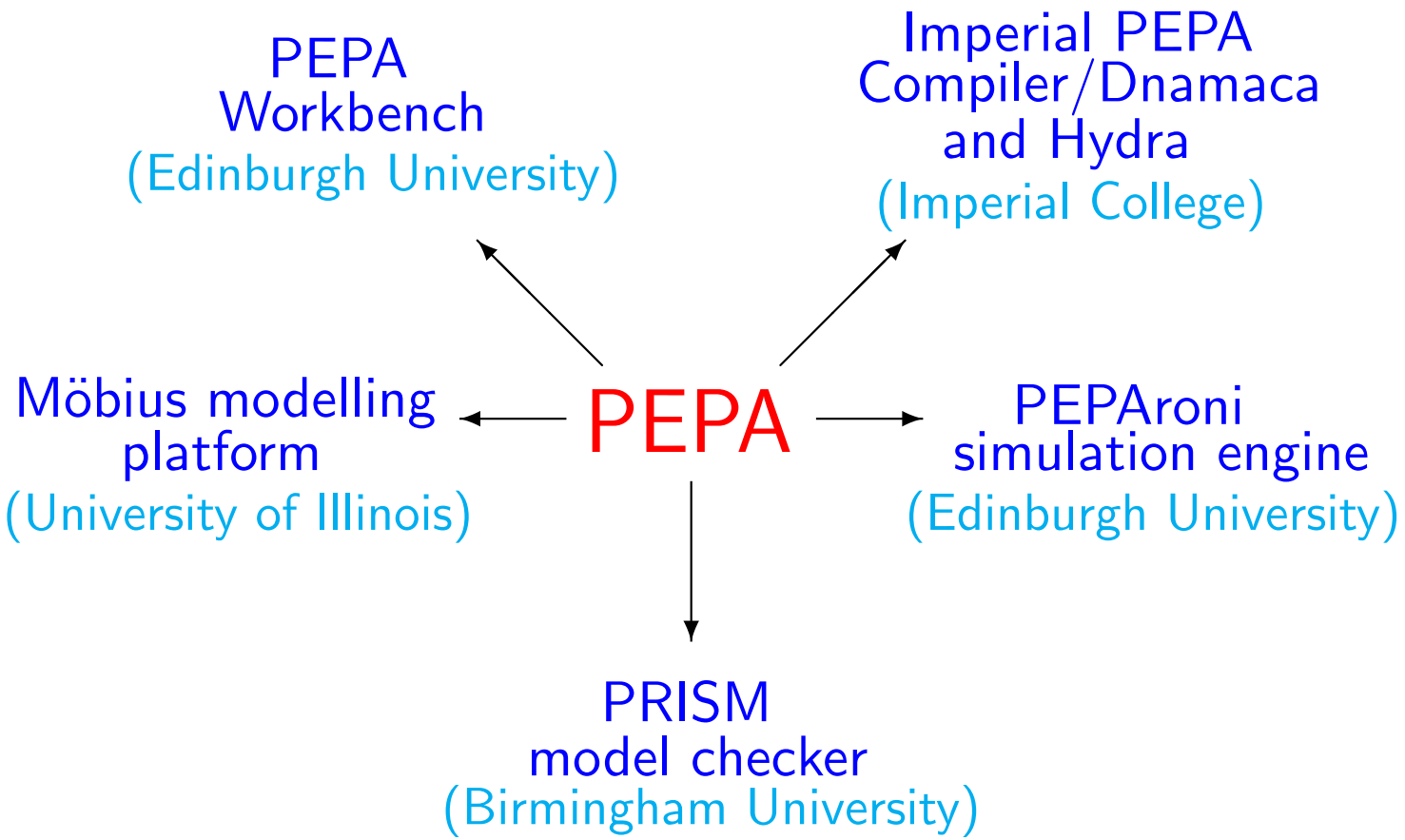


PEPA Case Studies (2)

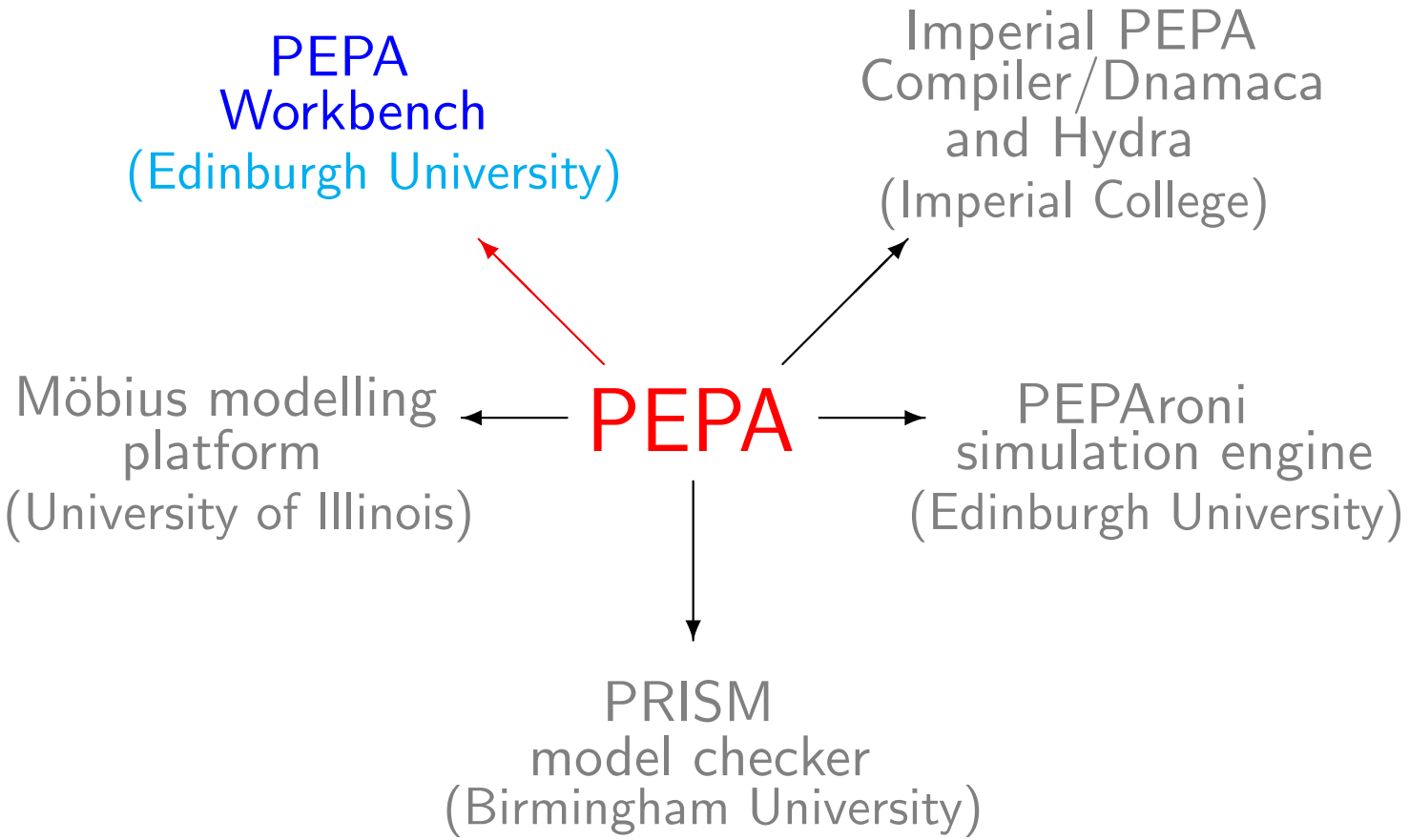
- Locks and movable bridges in inland shipping in Belgium (Knapen, Hasselt)
- Robotic workcells (Holton, Gilmore and Hillston, Bradford and Edinburgh)
- Cellular telephone networks (Kloul, Fourneau and Valois, Versailles)
- Automotive diagnostic expert systems (Console, Picardi and Ribaudo, Turin)



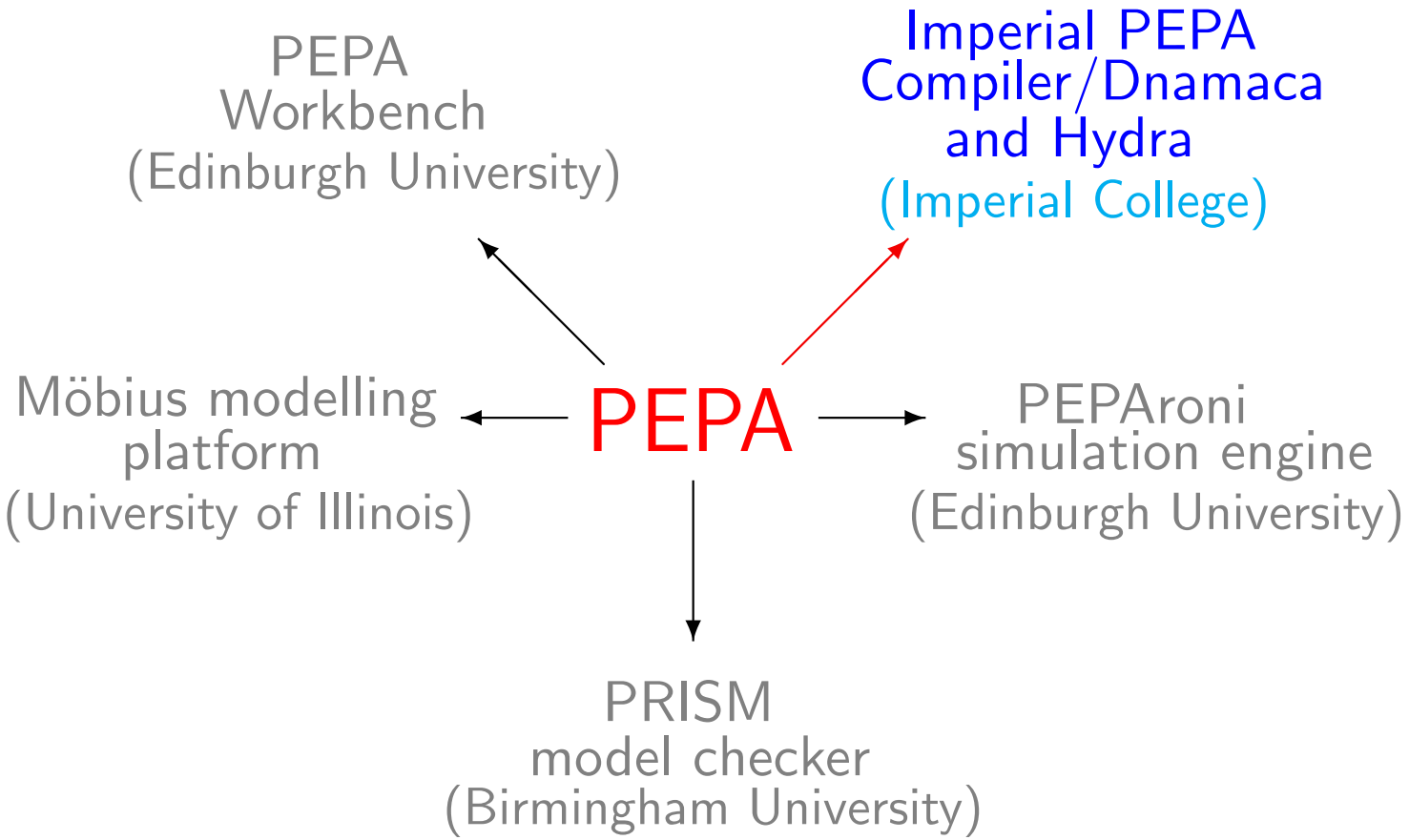
Tool Support



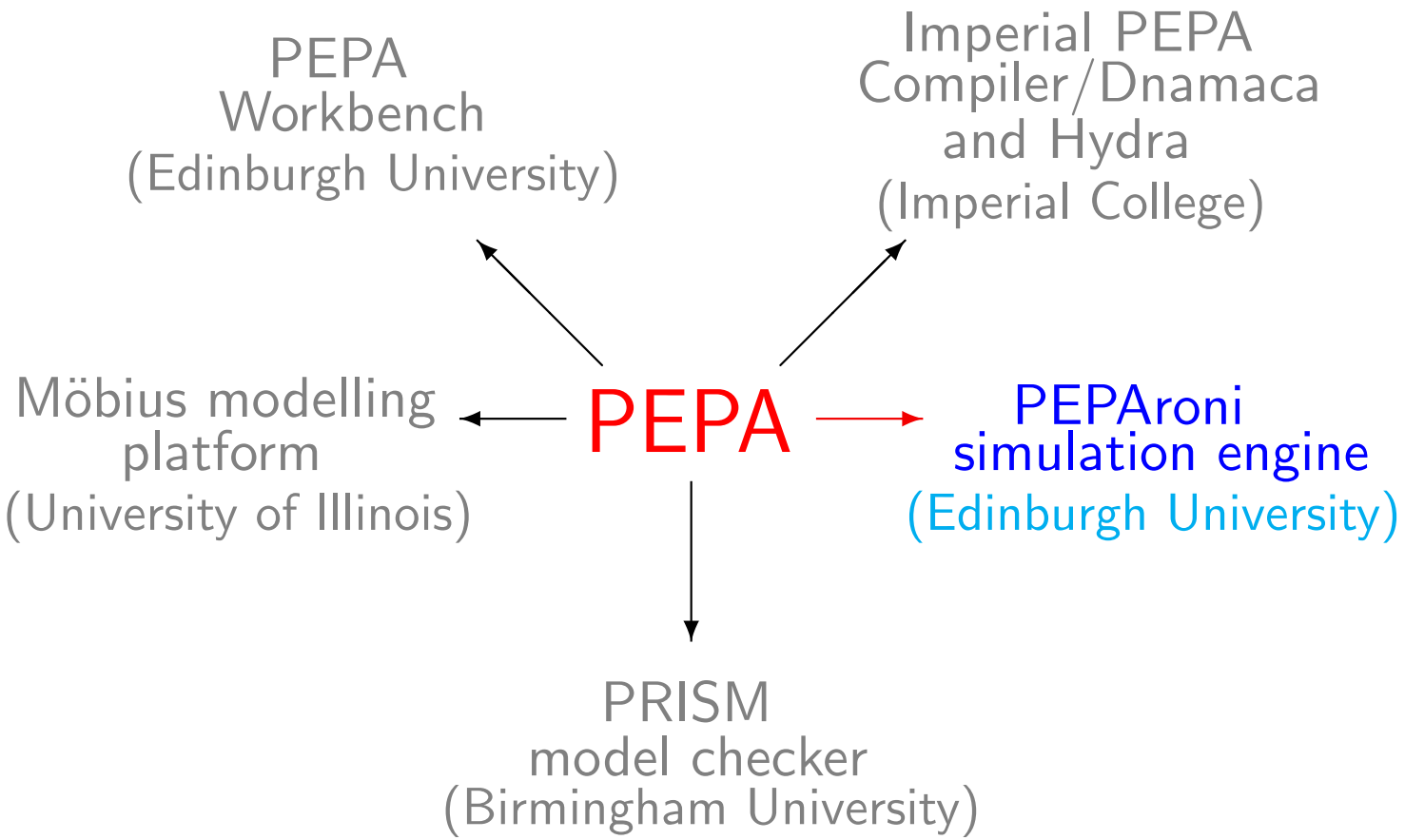
Tool Support



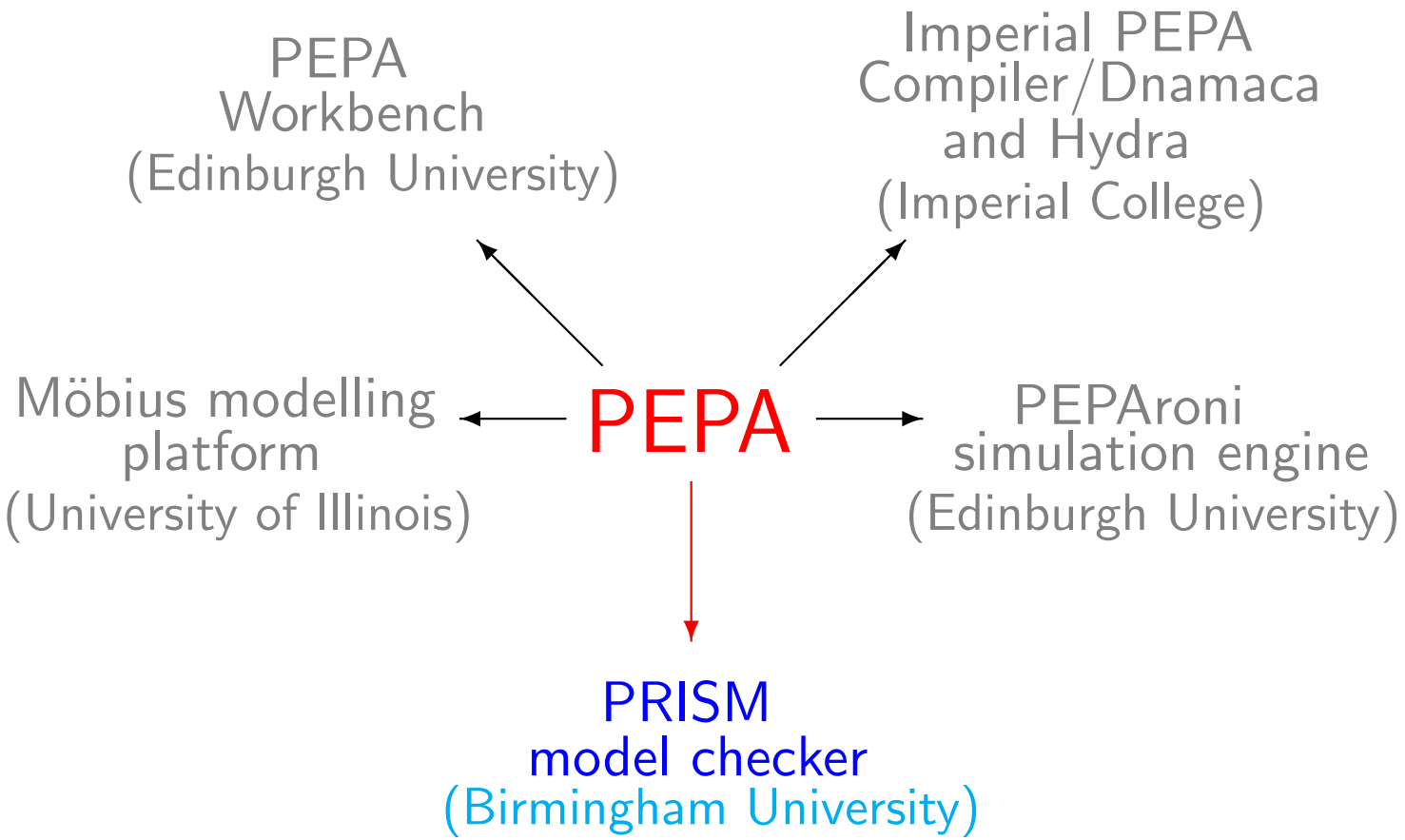
Tool Support



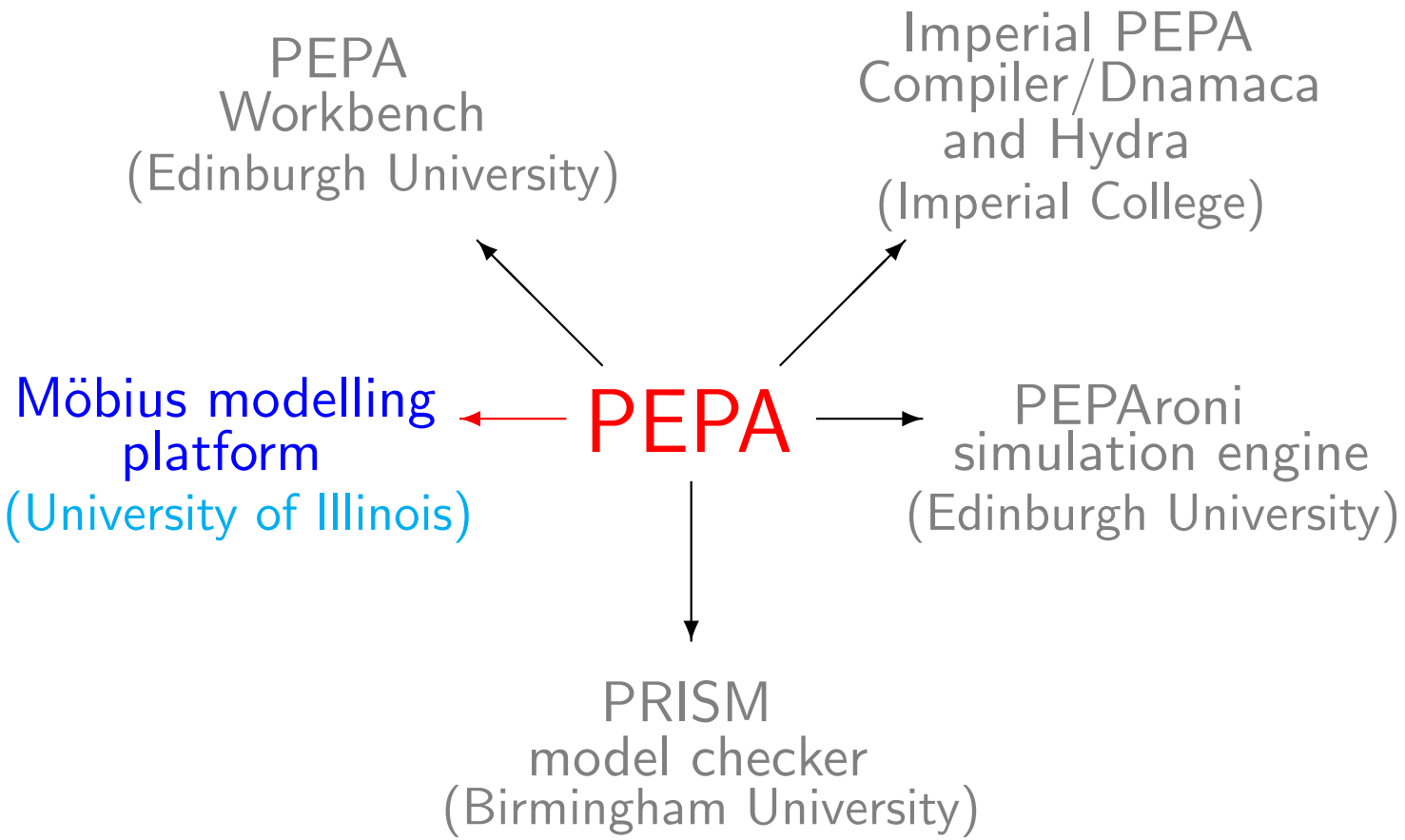
Tool Support



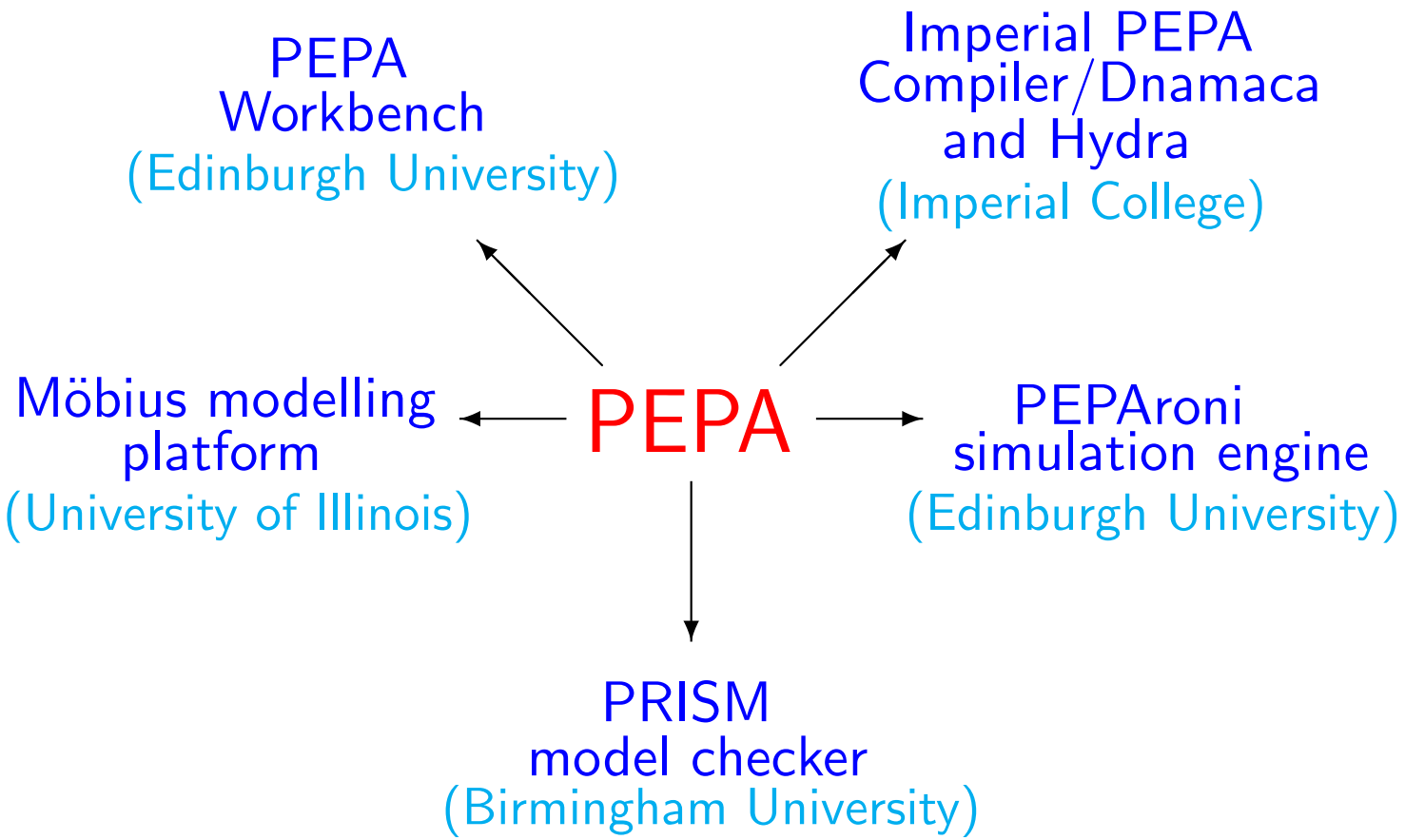
Tool Support



Tool Support



Tool Support



Roland the Gunslinger

- This sequence of small examples are based around a character called Roland Deschain.

Roland the Gunslinger

- This sequence of small examples are based around a character called Roland Deschain.
- Roland is a gunslinger and his life consists of wandering around firing his gun.

Roland the Gunslinger

- This sequence of small examples are based around a character called Roland Deschain.
- Roland is a gunslinger and his life consists of wandering around firing his gun.
- We will consider Roland in a number of different scenarios.

Roland the Gunslinger

- This sequence of small examples are based around a character called Roland Deschain.
- Roland is a gunslinger and his life consists of wandering around firing his gun.
- We will consider Roland in a number of different scenarios.
- These are not intended to be serious but they serve to

Roland the Gunslinger

- This sequence of small examples are based around a character called Roland Deschain.
- Roland is a gunslinger and his life consists of wandering around firing his gun.
- We will consider Roland in a number of different scenarios.
- These are not intended to be serious but they serve to
  - illustrate the main features of the language,

Roland the Gunslinger

- This sequence of small examples are based around a character called Roland Deschain.
- Roland is a gunslinger and his life consists of wandering around firing his gun.
- We will consider Roland in a number of different scenarios.
- These are not intended to be serious but they serve to
  - illustrate the main features of the language,
  - give you some experience of how models are constructed, and

Roland the Gunslinger

- This sequence of small examples are based around a character called Roland Deschain.
- Roland is a gunslinger and his life consists of wandering around firing his gun.
- We will consider Roland in a number of different scenarios.
- These are not intended to be serious but they serve to
  - illustrate the main features of the language,
  - give you some experience of how models are constructed, and
  - demonstrate a variety of solution techniques.

Roland alone

In the first scenario we consider Roland alone, with the single activity of firing his gun which is a six-shooter. When his gun is empty Roland will reload the gun and then continue shooting.

$$
\begin{aligned}
Roland_6 &\stackrel{def}{=} (fire, r_{fire}).Roland_5 \\
Roland_5 &\stackrel{def}{=} (fire, r_{fire}).Roland_4 \\
Roland_4 &\stackrel{def}{=} (fire, r_{fire}).Roland_3 \\
Roland_3 &\stackrel{def}{=} (fire, r_{fire}).Roland_2 \\
Roland_2 &\stackrel{def}{=} (fire, r_{fire}).Roland_1 \\
Roland_1 &\stackrel{def}{=} (fire, r_{fire}).Roland_{empty} \\
Roland_{empty} &\stackrel{def}{=} (reload, r_{reload}).Roland_6
\end{aligned}
$$

Roland with two guns

All self-respecting gun-slingers have one gun in each hand. A simplistic way to model this is two instances of *Roland* in parallel:

$$Roland_6 \parallel Roland_6$$

Roland with two guns

All self-respecting gun-slingers have one gun in each hand. A simplistic way to model this is two instances of *Roland* in parallel:

$$Roland_6 \parallel Roland_6$$

But this model does not capture the fact that Roland needs both hands in order to reload either gun. Thus we might assume that Roland only reloads both guns when both are empty.

$$Roland_6 \underset{\{reload\}}{\bowtie} Roland_6$$

Roland with two guns

All self-respecting gun-slingers have one gun in each hand. A simplistic way to model this is two instances of *Roland* in parallel:

$$Roland_6 \parallel Roland_6$$

But this model does not capture the fact that Roland needs both hands in order to reload either gun. Thus we might assume that Roland only reloads both guns when both are empty.

$$Roland_6 \underset{\{reload\}}{\bowtie} Roland_6$$

From now on we restrict Roland to his shotgun, which has two bullets and requires both hands for firing.

Roland meets an Enemy

- Upon his travels Roland encounters some enemies and when he does so he must fight them.
- Roland is the wildest gunslinger in the west so we assume that no enemy has the skill to seriously harm Roland.
- Each time Roland fires he might miss or hit his target.
- But with nothing to stop him he will keep firing until he successfully hits (and kills) the enemy.
- We assume that some sense of cowboy honour prevents any enemy attacking Roland if he is already involved in a gun fight.

The model

$$
\begin{aligned}
Roland_{idle} &\stackrel{def}{=} (attack, r_{attack}).Roland_2 \\
Roland_2 &\stackrel{def}{=} (hit, r_{hit}).(reload, r_{reload}).Roland_{idle} \\
&\quad + (miss, r_{miss}).Roland_1 \\
Roland_1 &\stackrel{def}{=} (hit, r_{hit}).(reload, r_{reload}).Roland_{idle} \\
&\quad + (miss, r_{miss}).Roland_{empty} \\
Roland_{empty} &\stackrel{def}{=} (reload, r_{reload}).Roland_2
\end{aligned}
$$

Parameter settings for the $Roland_2$ model

parameter	*value*	*explanation*
r_{fire}	1.0	Roland can fire the gun once per-second
$p_{hit\text{-}success}$	0.8	Roland has an 80% success rate
r_{hit}	0.8	$r_{fire} \times p_{hit\text{-}success}$
r_{miss}	0.2	$r_{fire} \times (1 - p_{hit\text{-}success})$
r_{reload}	0.3	It takes Roland about 3 seconds to reload
r_{attack}	0.01	Roland is attacked once every 100 seconds

Steady state analysis

We can calculate the probability that at arbitrary time Roland is involved in a battle based on the steady state probability that *Roland* is in any of the states in which a battle is on-going, i.e. $Roland_2$, $Roland_1$ and $Roland_{empty}$.

Steady state analysis

We can calculate the probability that at arbitrary time Roland is involved in a battle based on the steady state probability that *Roland* is in any of the states in which a battle is on-going, i.e. $Roland_2$, $Roland_1$ and $Roland_{empty}$.

Or we can calculate the probability that *Roland* is in the state $Roland_{idle}$ and subtract it from 1.

Steady state analysis

We can calculate the probability that at arbitrary time Roland is involved in a battle based on the steady state probability that *Roland* is in any of the states in which a battle is on-going, i.e. $Roland_2$, $Roland_1$ and $Roland_{empty}$.

Or we can calculate the probability that *Roland* is in the state $Roland_{idle}$ and subtract it from 1.

```
 State Measure 'roland peaceful'
mean                9.5490716180e-01
State Measure 'roland in battle'
mean                0.0450928382e-01
```

Passage-Time Analysis

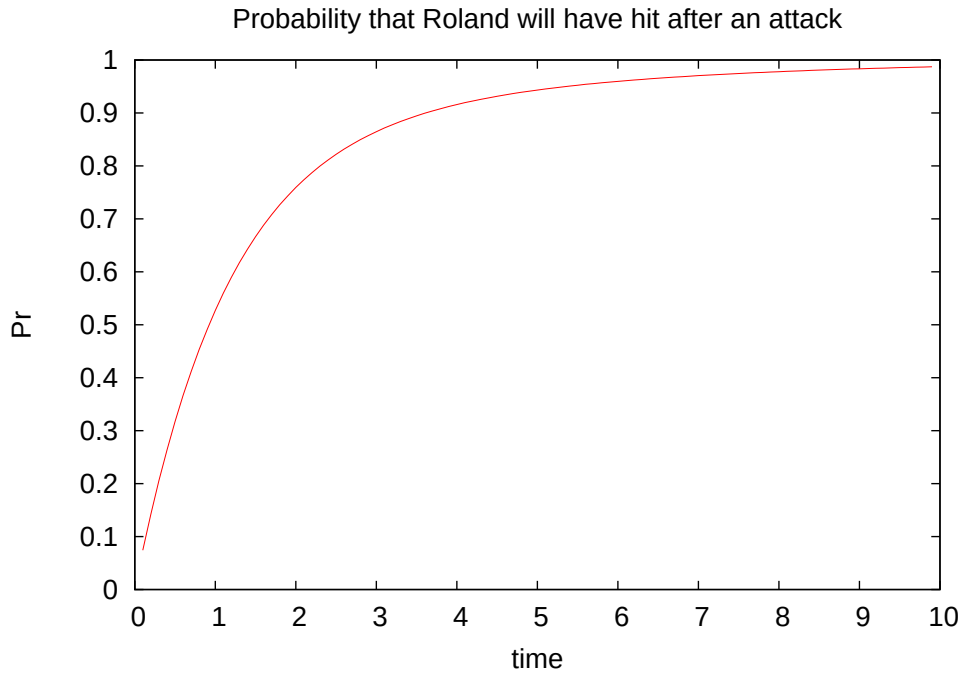
Passage-time analysis allows us to calculate measures such as the probability that Roland has killed his enemy at a given time after he is attacked.

Passage-Time Analysis

Passage-time analysis allows us to calculate measures such as the probability that Roland has killed his enemy at a given time after he is attacked.

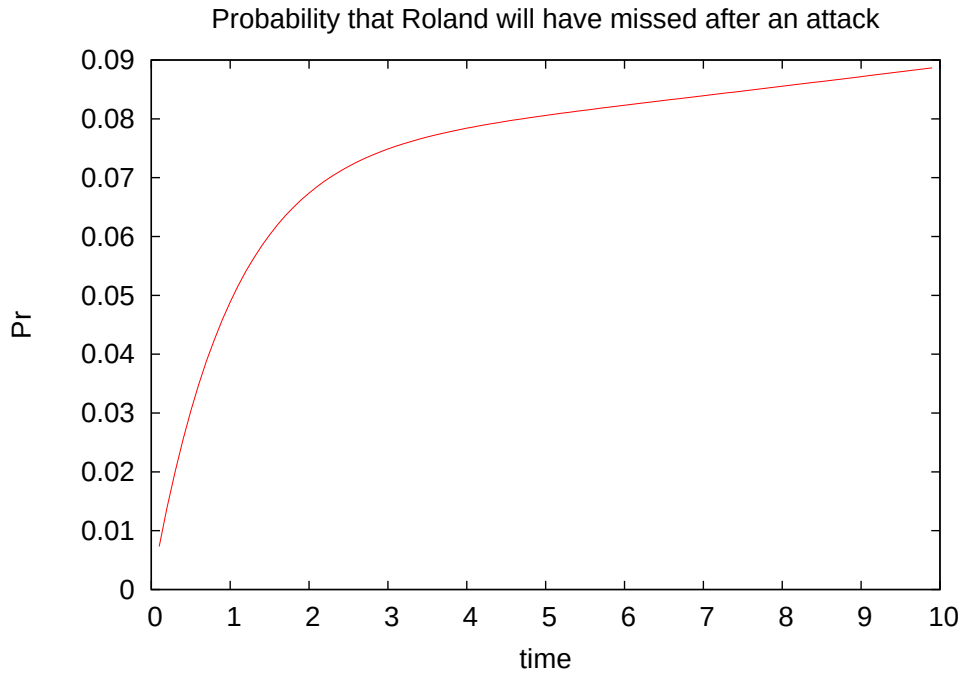
This would involve calculating the probability that the model performs a *hit* action within the given time after performing an *attack* action.

Passage-Time Analysis results



The probability that Roland will successfully perform a *hit* action a given time after an *attack*.

Passage-Time Analysis results



The probability that Roland has performed a *miss* action a given time after an *attack* action.

Cooperation

- In the previous model Roland's enemies were represented only implicitly.

Cooperation

- In the previous model Roland's enemies were represented only implicitly.
- We now consider a model in which the enemies appear explicitly and allow them to fight back.

Cooperation

- In the previous model Roland's enemies were represented only implicitly.
- We now consider a model in which the enemies appear explicitly and allow them to fight back.
- However for now we still assume that there are rather ineffectual and so they never seriously injure Roland.

Cooperation

- In the previous model Roland's enemies were represented only implicitly.
- We now consider a model in which the enemies appear explicitly and allow them to fight back.
- However for now we still assume that there are rather ineffectual and so they never seriously injure Roland.
- This model can be used to calculate properties such as the likelihood that an enemy will manage to fire one shot before they are killed by Roland.

Revised Model

$$
\begin{aligned}
Roland_{idle} &\stackrel{def}{=} (attack, \top).Roland_2 \\
Roland_2 &\stackrel{def}{=} (hit, r_{hit}).(reload, r_{reload}).Roland_{idle} \\
&\qquad + (miss, r_{miss}).Roland_1 \\
Roland_1 &\stackrel{def}{=} (hit, r_{hit}).(reload, r_{reload}).Roland_{idle} \\
&\qquad + (miss, r_{miss}).Roland_{empty} \\
Roland_{empty} &\stackrel{def}{=} (reload, r_{reload}).Roland_2 \\
\\
Enemies_{idle} &\stackrel{def}{=} (attack, r_{attack}).Enemies_{attack} \\
Enemies_{attack} &\stackrel{def}{=} (fire, r_{e\text{-}miss}).Enemies_{attack} \\
&\qquad + (hit, \top).Enemies_{idle} \\
\\
Roland_2 &\underset{\{hit\}}{\bowtie} Enemies_{idle}
\end{aligned}
$$

Additional parameters

parameter	*value*	*explanation*
r_{attack}	0.01	Roland is attacked once every 100 seconds
$r_{e\text{-}miss}$	0.3	Enemies can fire only once every 3 seconds

Levels of abstraction

- Notice that in this model the behaviour of the enemy has been simplified.

Levels of abstraction

- Notice that in this model the behaviour of the enemy has been simplified.
- There is no running out of bullets or reloading.

Levels of abstraction

- Notice that in this model the behaviour of the enemy has been simplified.
- There is no running out of bullets or reloading.
- This model can be thought of as an approximation to a more complicated component similar to the one which models Roland.

Levels of abstraction

- Notice that in this model the behaviour of the enemy has been simplified.
- There is no running out of bullets or reloading.
- This model can be thought of as an approximation to a more complicated component similar to the one which models Roland.
- Here the rate at which the enemy fires encompasses all of the actions, including the reloading of an empty gun.

Levels of abstraction

- Notice that in this model the behaviour of the enemy has been simplified.
- There is no running out of bullets or reloading.
- This model can be thought of as an approximation to a more complicated component similar to the one which models Roland.
- Here the rate at which the enemy fires encompasses all of the actions, including the reloading of an empty gun.
- We may choose to model a component in such an abstract way when the focus of our modelling is really elsewhere in the model.

Model Validation

It is also sometimes useful to carry out a validation of the model by calculating a metric which we believe we already know the value of.

Model Validation

It is also sometimes useful to carry out a validation of the model by calculating a metric which we believe we already know the value of.

For example in this model we could make such a sanity check by calculating the probability that the model is in a state in which Roland is idle but the enemies are not, or vice versa.

Model Validation

It is also sometimes useful to carry out a validation of the model by calculating a metric which we believe we already know the value of.

For example in this model we could make such a sanity check by calculating the probability that the model is in a state in which Roland is idle but the enemies are not, or vice versa.

This should never occur and hence the probability should be zero.

Sensitivity Analysis

- Sensitivity analysis studies how much influence particular parameter values, such as activity rates, have on performance metrics calculated for the system as a whole.

Sensitivity Analysis

- Sensitivity analysis studies how much influence particular parameter values, such as activity rates, have on performance metrics calculated for the system as a whole.
- A single activity in a PEPA model may have a significant impact on the dynamics of the model, or, conversely, may exert very little influence.

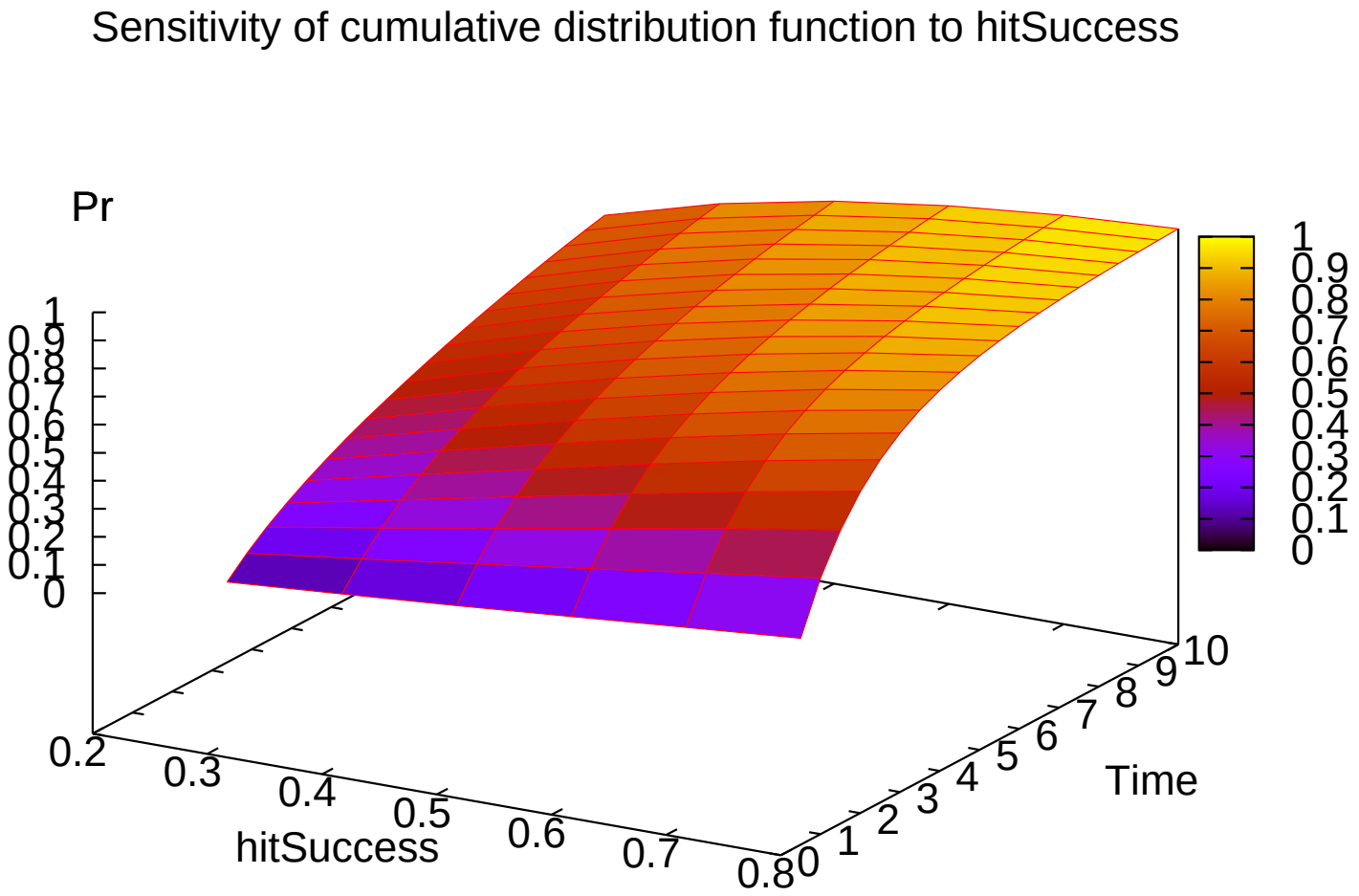
Sensitivity Analysis

- Sensitivity analysis studies how much influence particular parameter values, such as activity rates, have on performance metrics calculated for the system as a whole.
- A single activity in a PEPA model may have a significant impact on the dynamics of the model, or, conversely, may exert very little influence.
- Sensitivity analysis is performed by solving the model many times while varying the rates slightly.

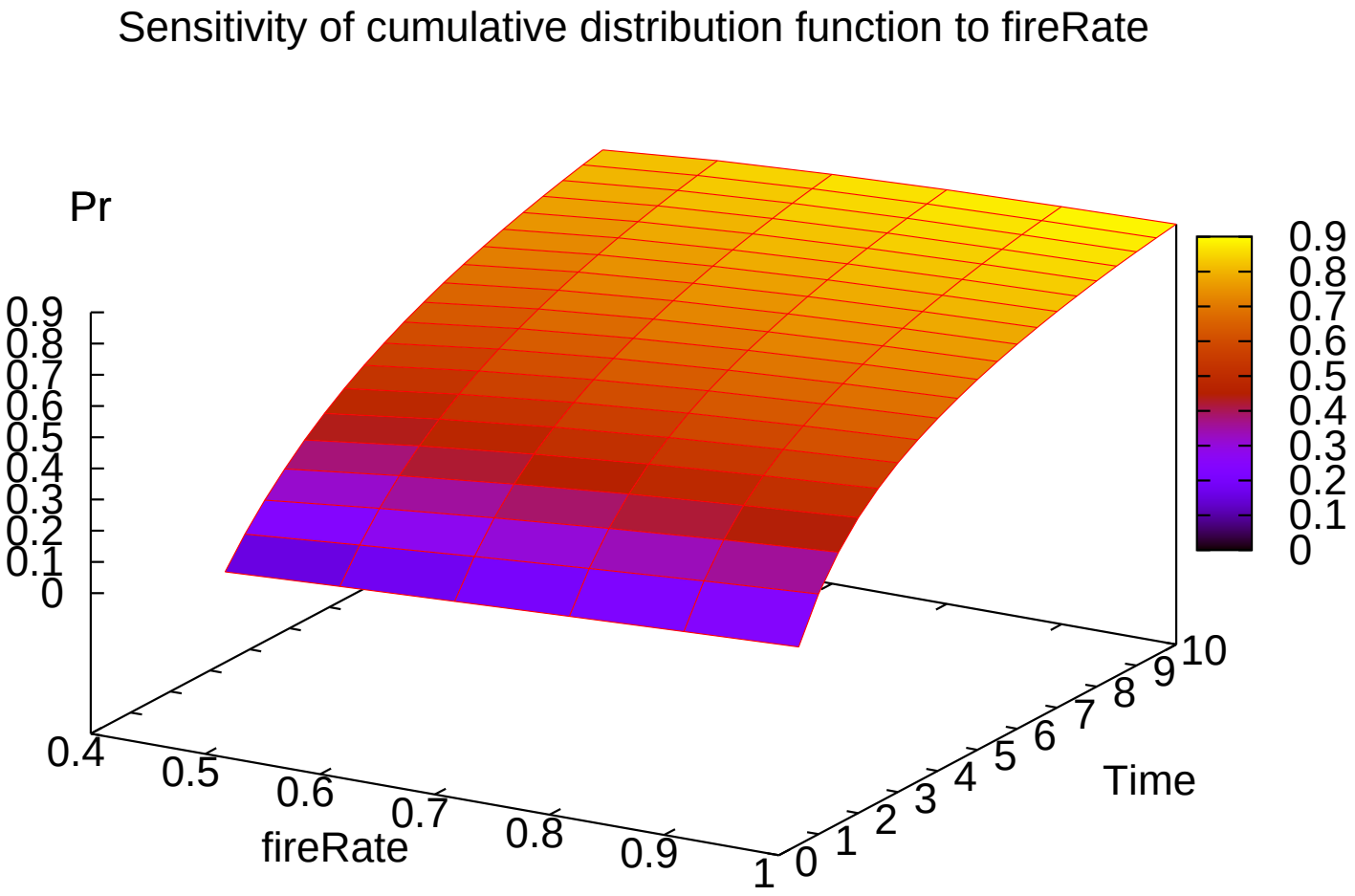
Sensitivity Analysis

- Sensitivity analysis studies how much influence particular parameter values, such as activity rates, have on performance metrics calculated for the system as a whole.
- A single activity in a PEPA model may have a significant impact on the dynamics of the model, or, conversely, may exert very little influence.
- Sensitivity analysis is performed by solving the model many times while varying the rates slightly.
- For this model we chose to vary three of the rates involved and measured the passage time between an *attack* and a *hit* activity, for each combination of rates.

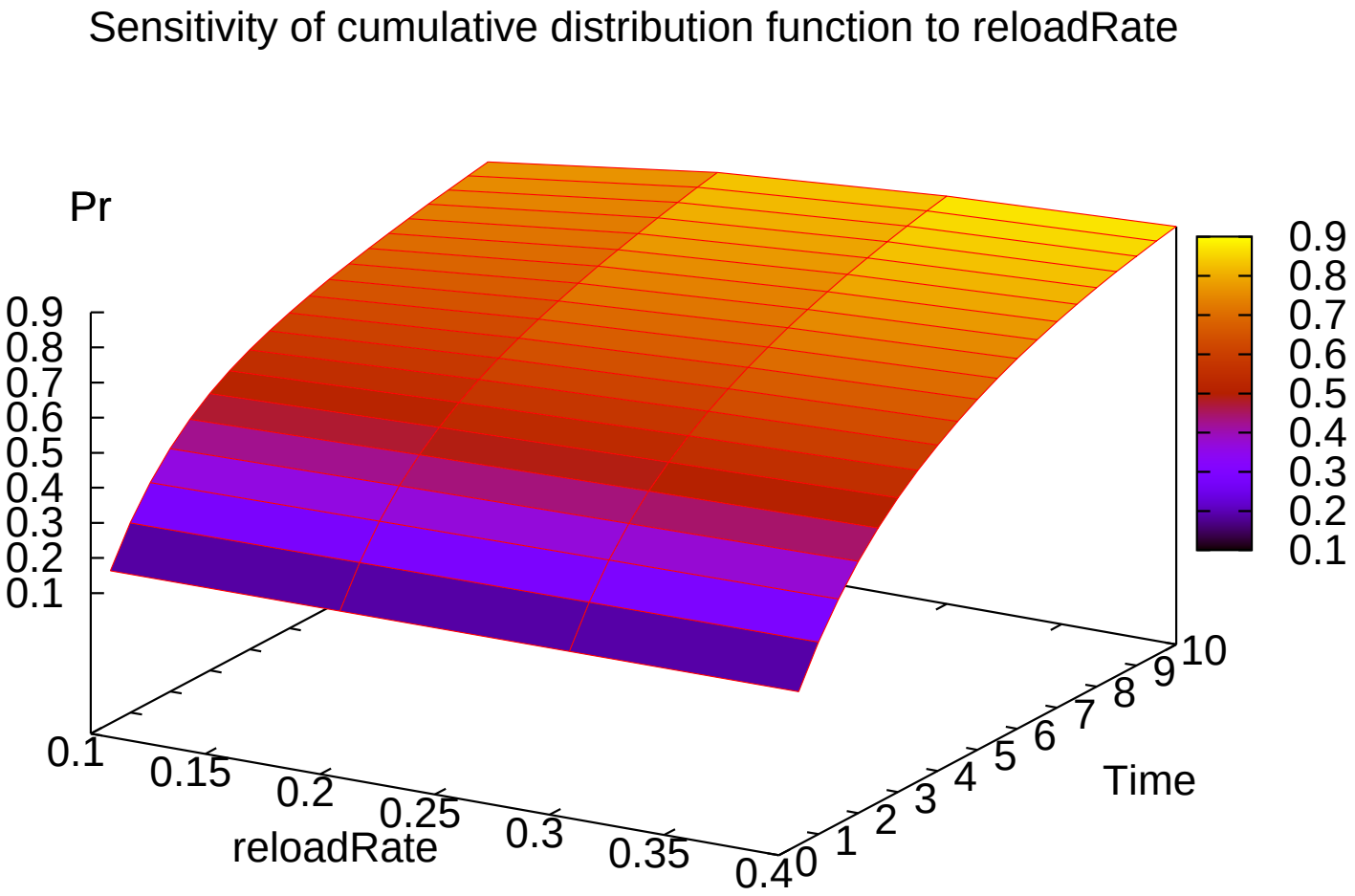
Sensitivity Analysis: Results



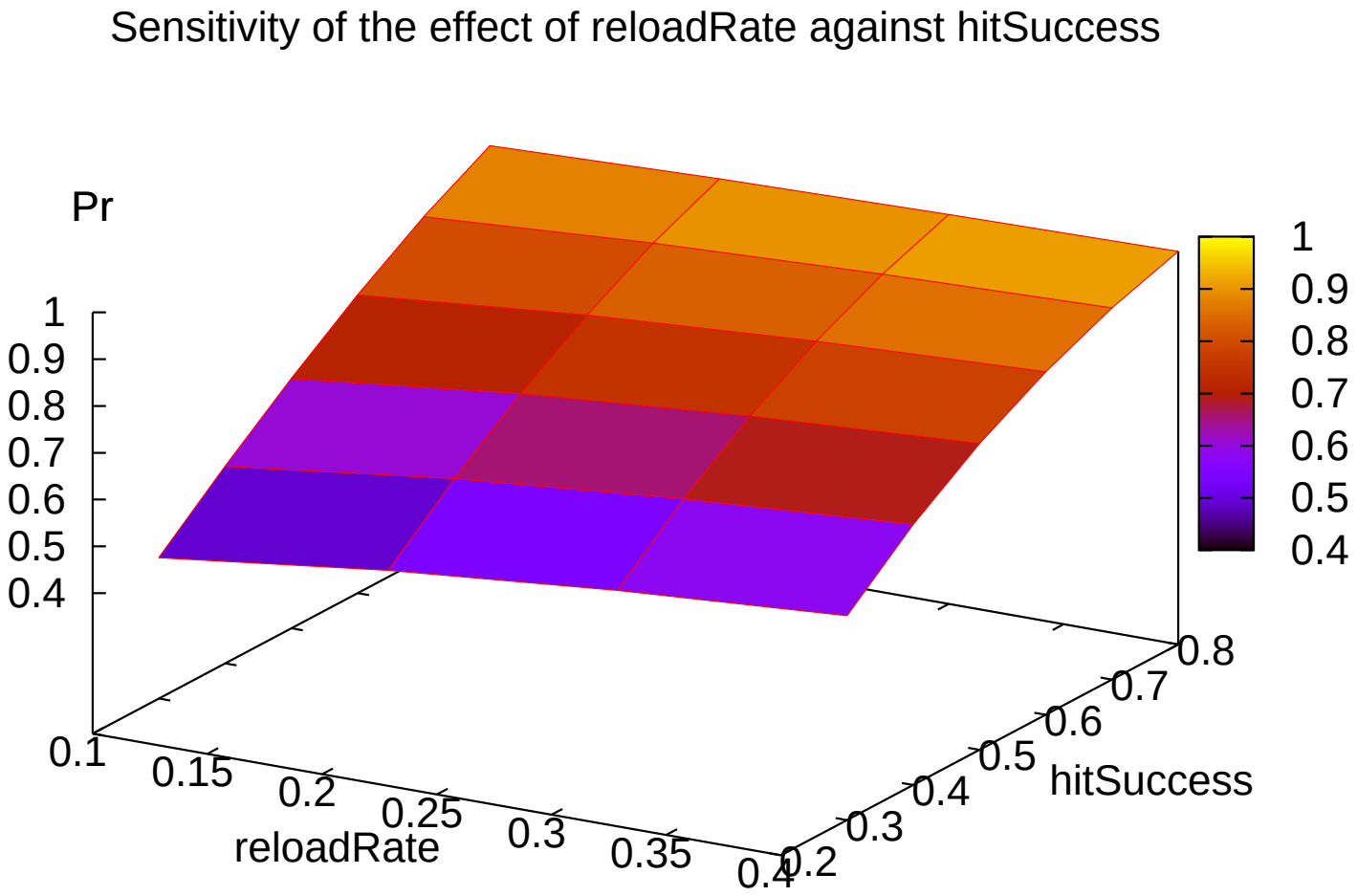
Sensitivity Analysis: Results



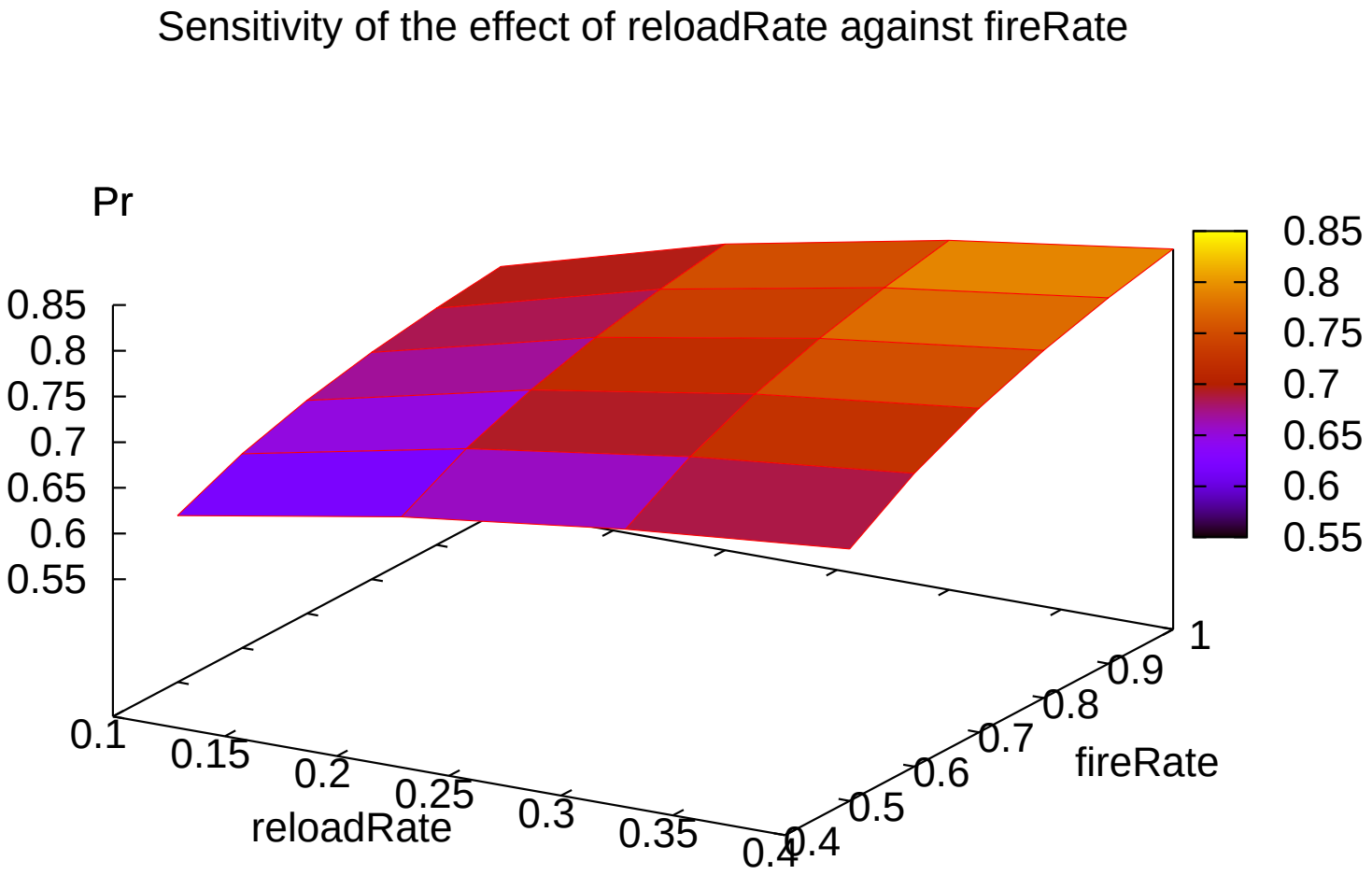
Sensitivity Analysis: Results



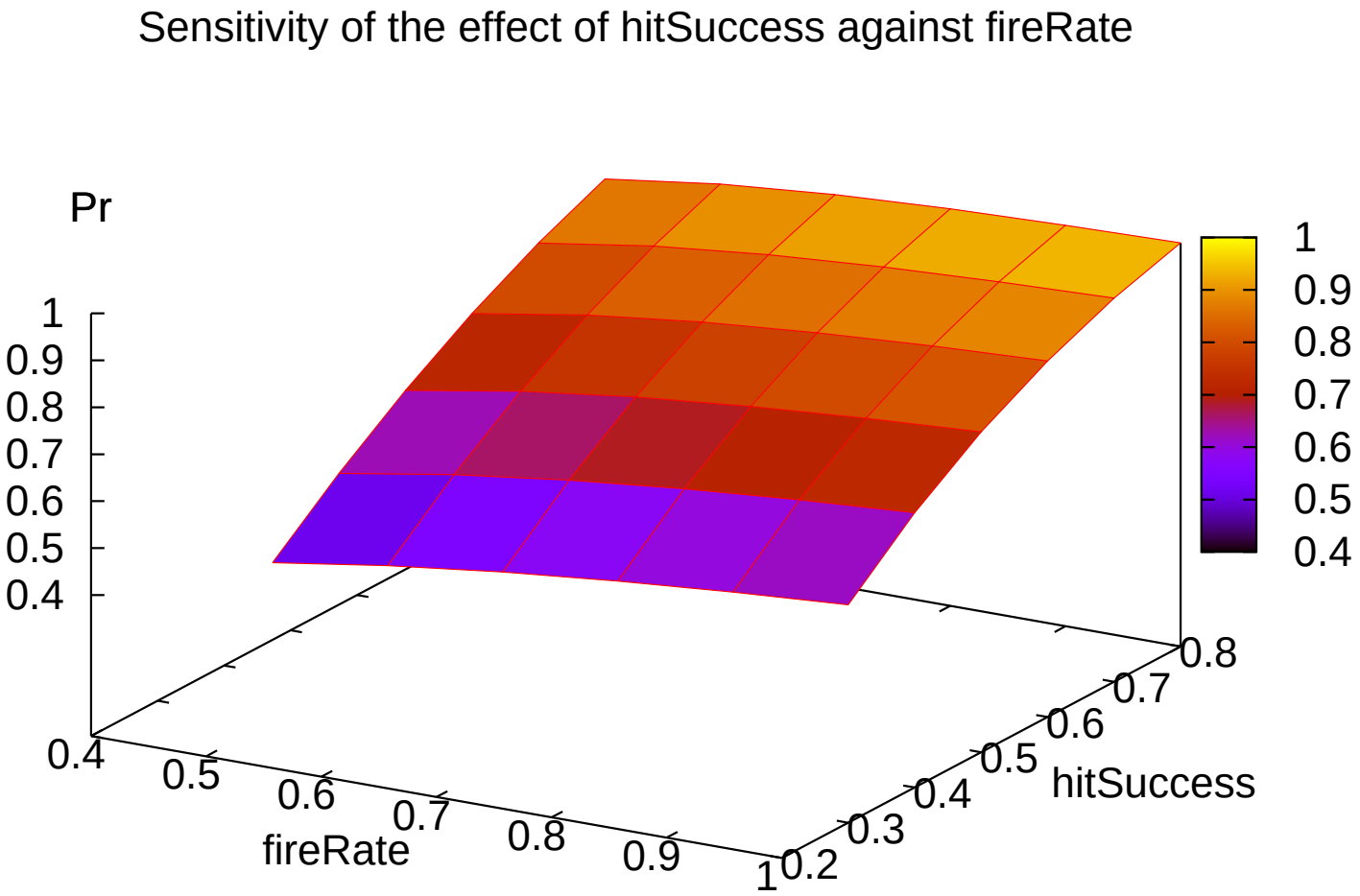
Sensitivity Analysis: Results



Sensitivity Analysis: Results



Sensitivity Analysis: Results



Accurate Enemies

- We now allow the enemies of Roland to actually hit him. This means that Roland may die. It is important to note that this has the consequence that the model will always deadlock. The underlying Markov process is no longer ergodic.

Accurate Enemies

- We now allow the enemies of Roland to actually hit him. This means that Roland may die. It is important to note that this has the consequence that the model will always deadlock. The underlying Markov process is no longer ergodic.
- We assume that the enemies can only hit Roland once every 50 seconds. This rate approximates the rate of a more detailed model in which we would assign a process to the enemies which is much like that of the process which describes Roland.

Accurate Enemies

- We now allow the enemies of Roland to actually hit him. This means that Roland may die. It is important to note that this has the consequence that the model will always deadlock. The underlying Markov process is no longer ergodic.
- We assume that the enemies can only hit Roland once every 50 seconds. This rate approximates the rate of a more detailed model in which we would assign a process to the enemies which is much like that of the process which describes Roland.
- The only new parameter is $r_{e\text{-}hit}$ which is assigned a value 0.02 to reflect this assumption.

New Roland

$$
\begin{aligned}
Roland_{idle} &\stackrel{def}{=} (attack, \top).Roland_2 \\
Roland_2 &\stackrel{def}{=} (hit, r_{hit}).(reload, r_{reload}).Roland_{idle} \\
&+ (miss, r_{miss}).Roland_1 \\
&+ (e\text{-}hit, \top).Roland_{dead} \\
Roland_1 &\stackrel{def}{=} (hit, r_{hit}).(reload, r_{reload}).Roland_{idle} \\
&+ (miss, r_{miss}).Roland_{empty} \\
&+ (e\text{-}hit, \top).Roland_{dead} \\
Roland_{empty} &\stackrel{def}{=} (reload, r_{reload}).Roland_2 \\
&+ (e\text{-}hit, \top).Roland_{dead} \\
Roland_{dead} &\stackrel{def}{=} Stop
\end{aligned}
$$

New Enemy

$$
\begin{aligned}
Enemies_{idle} &\stackrel{def}{=} (attack, r_{attack}).Enemies_{attack} \\
Enemies_{attack} &\stackrel{def}{=} (e\text{-}hit, r_{e\text{-}hit}).Enemies_{idle} \\
&\quad + \quad (hit, \top).Enemies_{idle}
\end{aligned}
$$

$$
Roland_{idle} \underset{\{hit, attack, e\text{-}hit\}}{\bowtie} Enemies_{idle}
$$

Model Analysis

Steady-State Analysis Since there is an infinite supply of enemies eventually Roland will always die and the model will deadlock.

Model Analysis

Steady-State Analysis Since there is an infinite supply of enemies eventually Roland will always die and the model will deadlock.

Transient Analysis Transient analysis on this model can be used to calculate the probability that Roland is dead after a given amount of time. As time increases this should tend towards probability 1.

Model Analysis

Steady-State Analysis Since there is an infinite supply of enemies eventually Roland will always die and the model will deadlock.

Transient Analysis Transient analysis on this model can be used to calculate the probability that Roland is dead after a given amount of time. As time increases this should tend towards probability 1.

Passage-Time Analysis Passage-time analysis could be used to calculate the probability of a given event happening at a given time after another given event, e.g. from an attack on Roland until he dies or wins the gun fight.

Roland makes a friend

In the next revision of the model we introduce an accomplice who is befriended by Roland and who, when Roland is attacked, fights alongside him.

Roland makes a friend

In the next revision of the model we introduce an accomplice who is befriended by Roland and who, when Roland is attacked, fights alongside him.

In this scenario cooperation is used to synchronise between components of the model such that they observe events which they neither directly cause nor are directly affected by.

Roland makes a friend

In the next revision of the model we introduce an accomplice who is befriended by Roland and who, when Roland is attacked, fights alongside him.

In this scenario cooperation is used to synchronise between components of the model such that they observe events which they neither directly cause nor are directly affected by.

Whenever either Roland or the accomplice kills the enemy the other must witness this action, so as to stop firing at a dead opponent (it would be a waste of ammunition!).

A new component for Roland

$$
\begin{aligned}
Roland_{idle} &\stackrel{def}{=} (attack, \top).Roland_2 \\
&+ (befriend, r_{befriend}).Roland_{idle} \\
Roland_2 &\stackrel{def}{=} (hit, r_{hit}).Roland_{hit} + (miss, r_{miss}).Roland_1 \\
&+ (a\text{-}hit, \top).Roland_{idle} \\
Roland_1 &\stackrel{def}{=} (hit, r_{hit}).Roland_{hit} \\
&+ (miss, r_{miss}).Roland_{empty} \\
&+ (a\text{-}hit, \top).Roland_{idle} \\
Roland_{hit} &\stackrel{def}{=} (reload, r_{reload}).Roland_{idle} \\
Roland_{empty} &\stackrel{def}{=} (reload, r_{reload}).Roland_2 \\
&+ (a\text{-}hit, \top).Roland_{hit}
\end{aligned}
$$

Synchronising Roland and the Accomplice

- When there is an accomplice, he and Roland fight together against the enemy — this involves some cooperation.

Synchronising Roland and the Accomplice

- When there is an accomplice, he and Roland fight together against the enemy — this involves some cooperation.
- This could leave Roland vulnerable when there is no accomplice present.
- To prevent this we introduce a dummy component representing the absence of an accomplice.

Synchronising Roland and the Accomplice

- When there is an accomplice, he and Roland fight together against the enemy — this involves some cooperation.
- This could leave Roland vulnerable when there is no accomplice present.
- To prevent this we introduce a dummy component representing the absence of an accomplice.

Synchronising Roland and the Accomplice

- When there is an accomplice, he and Roland fight together against the enemy — this involves some cooperation.
- This could leave Roland vulnerable when there is no accomplice present.
- To prevent this we introduce a dummy component representing the absence of an accomplice.
- In this state the accomplice component will passively participate in any attack which Roland makes.

Synchronising Roland and the Accomplice

- When there is an accomplice, he and Roland fight together against the enemy — this involves some cooperation.
- This could leave Roland vulnerable when there is no accomplice present.
- To prevent this we introduce a dummy component representing the absence of an accomplice.
- In this state the accomplice component will passively participate in any attack which Roland makes.

$$\begin{aligned} Acmpl_{abs} &\stackrel{def}{=} (befriend, r_{befriend}).Acmpl_{idle} \\ &+ (hit, \top).Acmpl_{abs} + (attack, \top).Acmpl_{abs} \end{aligned}$$

Component for the Accomplice

$$
\begin{aligned}
Acmpl_{idle} &\stackrel{def}{=} (attack, \top).Acmpl_2 \\
Acmpl_2 &\stackrel{def}{=} (a\text{-}hit, r_{a\text{-}hit}).Acmpl_{hit} + (hit, \top).Acmpl_{idle} \\
&\quad + (miss, r_{miss}).Acmpl_1 + (enemy\text{-}hit, \top).Acmpl_{abs} \\
Acmpl_1 &\stackrel{def}{=} (a\text{-}hit, r_{a\text{-}hit}).Acmpl_{hit} + (hit, \top).Acmpl_{hit} \\
&\quad + (miss, r_{miss}).Acmpl_{empty} + (enemy\text{-}hit, \top).Acmpl_{abs} \\
Acmpl_{hit} &\stackrel{def}{=} (reload, r_{a\text{-}reload}).Acmpl_{idle} \\
Acmpl_{empty} &\stackrel{def}{=} (reload, r_{a\text{-}reload}).Acmpl_2 + (hit, \top).Acmpl_{hit}
\end{aligned}
$$

Parameter Settings for the Accomplice

parameter	*value*	*explanation*
$r_{befriend}$	0.001	Roland befriends a stranger once every 1000 seconds
$r_{a\text{-}fire}$	1.0	the accomplice can also fire once per second
$p_{a\text{-}hit\text{-}success}$	0.6	the accomplice has a 60 percent accuracy
$r_{a\text{-}hit}$	0.6	$r_{fire} \times p_{hit\text{-}success}$
$r_{a\text{-}miss}$	0.4	$r_{fire} \times (1.0 - p_{hit\text{-}success})$
$r_{a\text{-}reload}$	0.25	it takes the accomplice 4 seconds to reload

Component for the Enemy

The component representing the enemy is similar to before.

$$
\begin{aligned}
Enemies_{idle} &\stackrel{def}{=} (attack, r_{attack}).Enemies_{attack} \\
Enemies_{attack} &\stackrel{def}{=} (enemy\text{-}hit, r_{e\text{-}hit}).Enemies_{attack} \\
&+ (hit, \top).(enemy\text{-}die, r_{e\text{-}die}).Enemies_{idle} \\
&+ (a\text{-}hit, \top).(enemy\text{-}die, r_{e\text{-}die}).Enemies_{idle}
\end{aligned}
$$

The system equation is as follows:

$$
(Roland_{idle} \underset{\{attack, hit, a\text{-}hit, befriend\}}{\bowtie} Acmpl_{abs}) \underset{\{attack, hit, a\text{-}hit, enemy\text{-}hit\}}{\bowtie} Enemies_{idle}
$$

Model Analysis

Steady-State Analysis

- As before we can determine the probability that Roland is involved in a gun battle at an arbitrary time.

Model Analysis

Steady-State Analysis

- As before we can determine the probability that Roland is involved in a gun battle at an arbitrary time.
- We could also determine the likelihood that Roland has an accomplice at an arbitrary time.

Model Analysis

Steady-State Analysis

- As before we can determine the probability that Roland is involved in a gun battle at an arbitrary time.
- We could also determine the likelihood that Roland has an accomplice at an arbitrary time.
- Since Roland cannot perform a befriending action while currently involved in a battle, the probabilty that Roland is in such a battle clearly affects the probability that he is alone in his quest.

Model Analysis

Steady-State Analysis

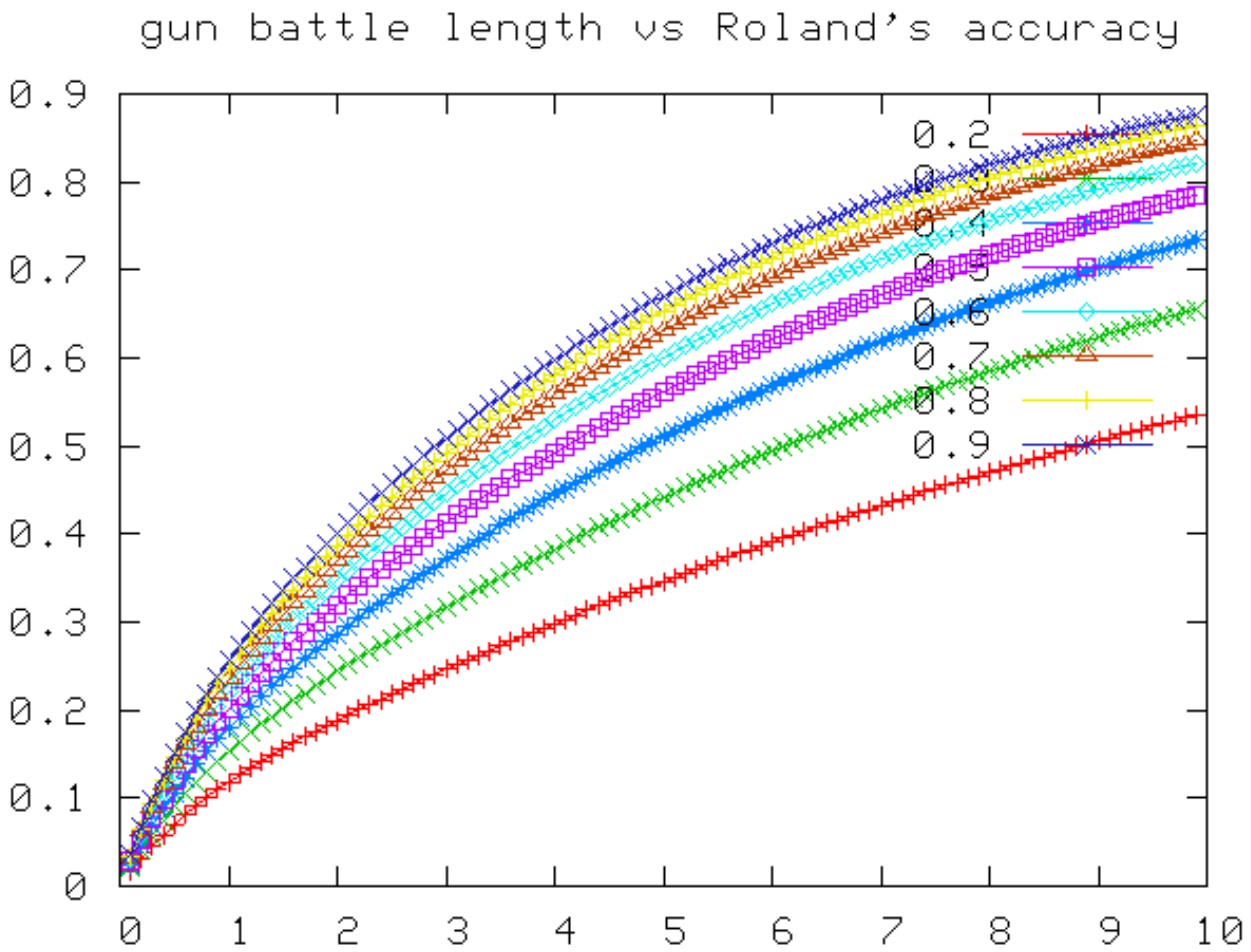
- As before we can determine the probability that Roland is involved in a gun battle at an arbitrary time.
- We could also determine the likelihood that Roland has an accomplice at an arbitrary time.
- Since Roland cannot perform a befriending action while currently involved in a battle, the probabilty that Roland is in such a battle clearly affects the probability that he is alone in his quest.
- For example, if Roland's success rate is reduced, gun battles will take longer to resolve and Roland will be involved in a gun battle more often. Consequently he will befriend fewer accomplices.

Model Analysis

Transient Analysis

An example transient analysis would be to determine the expected time after Roland has set off before he meets his first accomplice.

Model Analysis



Model Analysis

Passage-Time Analysis

- An example analysis would be to calculate the passage-time from an *attack* action until the death of the enemy or of the accomplice.

Model Analysis

Passage-Time Analysis

- An example analysis would be to calculate the passage-time from an *attack* action until the death of the enemy or of the accomplice.
- Since all gun battles now end in the enemy being killed stopping the analysis there would give us the expected duration of any one gun battle.

Model Analysis

Passage-Time Analysis

- An example analysis would be to calculate the passage-time from an *attack* action until the death of the enemy or of the accomplice.
- Since all gun battles now end in the enemy being killed stopping the analysis there would give us the expected duration of any one gun battle.
- There is also the possibility to start the analysis from the *befriend* action and stop it with the death of the accomplice.

Hiding

- Currently there is nothing in the model to stop an enemy from disrupting the interaction between Roland and his accomplice, e.g. by performing a *befriend* action.

Hiding

- Currently there is nothing in the model to stop an enemy from disrupting the interaction between Roland and his accomplice, e.g. by performing a *befriend* action.
- One way to avoid this is to 'hide' those actions only Roland and the accomplice should cooperate on.

Hiding

- Currently there is nothing in the model to stop an enemy from disrupting the interaction between Roland and his accomplice, e.g. by performing a *befriend* action.
- One way to avoid this is to 'hide' those actions only Roland and the accomplice should cooperate on.
- To do this for our model we can simply change the system equation:

$$((Roland_{idle} \underset{L_1}{\bowtie} Acmpl)/\{befriend\}) \underset{L_2}{\bowtie} Enemies_{idle}$$

where $L_1 = \{attackhit, a\text{-}hit, befriend\}$ and $L_2 = \{attack, hit, a\text{-}hit, enemy\text{-}hit\}$.

Outline

SPA Tool Support

SPA Tool Support

Several software tools supporting Stochastic Process Algebras have been developed over the years:

SPA Tool Support

Several software tools supporting Stochastic Process Algebras have been developed over the years:

- MoDeST

SPA Tool Support

Several software tools supporting Stochastic Process Algebras have been developed over the years:

- MoDeST
- TIPP-Tool

SPA Tool Support

Several software tools supporting Stochastic Process Algebras have been developed over the years:

- MoDeST
- TIPP-Tool
- TwoTowers

PEPA Tool Support

PEPA Tool Support

PEPA is amenable to several analysis techniques through a number of supporting tools.

PEPA Tool Support

PEPA is amenable to several analysis techniques through a number of supporting tools.

- The PEPA Workbench offers support for Markovian steady-state analysis, allowing computation of performance measures such as throughput and utilisation.

PEPA Tool Support

PEPA is amenable to several analysis techniques through a number of supporting tools.

- The PEPA Workbench offers support for Markovian steady-state analysis, allowing computation of performance measures such as throughput and utilisation.
- The Imperial PEPA Compiler translates PEPA models into the input format for Dnamaca, providing both steady-state and passage time analysis.

PEPA Tool Support

PEPA is amenable to several analysis techniques through a number of supporting tools.

- The PEPA Workbench offers support for Markovian steady-state analysis, allowing computation of performance measures such as throughput and utilisation.
- The Imperial PEPA Compiler translates PEPA models into the input format for Dnamaca, providing both steady-state and passage time analysis.
- Model checking via Continuous Stochastic Logic is available in PRISM which has built-in support for PEPA.

PEPA Tool Support

PEPA is amenable to several analysis techniques through a number of supporting tools.

- The PEPA Workbench offers support for Markovian steady-state analysis, allowing computation of performance measures such as throughput and utilisation.
- The Imperial PEPA Compiler translates PEPA models into the input format for Dnamaca, providing both steady-state and passage time analysis.
- Model checking via Continuous Stochastic Logic is available in PRISM which has built-in support for PEPA.
- PEPA has been integrated into the Möbius multi-paradigm modelling tool.

PEPA Tool Support

PEPA is amenable to several analysis techniques through a number of supporting tools.

- The PEPA Workbench offers support for Markovian steady-state analysis, allowing computation of performance measures such as throughput and utilisation.
- The Imperial PEPA Compiler translates PEPA models into the input format for Dnamaca, providing both steady-state and passage time analysis.
- Model checking via Continuous Stochastic Logic is available in PRISM which has built-in support for PEPA.
- PEPA has been integrated into the Möbius multi-paradigm modelling tool.
- The PEPA Plug-in Project permits Markovian steady-state analysis and stochastic simulation.

The PEPA Plug-in Project

The PEPA Plug-in Project

- The PEPA Plug-in Project is built on top of the Eclipse technology and it is deployed as a collection of plug-ins for the Eclipse IDE released under GPL.

The PEPA Plug-in Project

- The PEPA Plug-in Project is built on top of the Eclipse technology and it is deployed as a collection of plug-ins for the Eclipse IDE released under GPL.
- It has been successfully tested on Eclipse 3.2 running on various Linux distributions, Mac OS X, and Windows XP

The PEPA Plug-in Project

- The PEPA Plug-in Project is built on top of the Eclipse technology and it is deployed as a collection of plug-ins for the Eclipse IDE released under GPL.
- It has been successfully tested on Eclipse 3.2 running on various Linux distributions, Mac OS X, and Windows XP
- The tool is available for download at:
  `http://homepages.inf.ed.ac.uk/mtribast/`

The PEPA Plug-in Project

- The PEPA Plug-in Project is built on top of the Eclipse technology and it is deployed as a collection of plug-ins for the Eclipse IDE released under GPL.
- It has been successfully tested on Eclipse 3.2 running on various Linux distributions, Mac OS X, and Windows XP
- The tool is available for download at:
  `http://homepages.inf.ed.ac.uk/mtribast/`
- The examples discussed throughout this presentation are available at the PEPA Home Page:
  `http://www.dcs.ed.ac.uk/pepa/`

Key Plug-ins

Key Plug-ins

- PEPAto provides core services for PEPA. It can be used as a library by third-party applications.

Key Plug-ins

- PEPAto provides core services for PEPA. It can be used as a library by third-party applications.
- Eclipse Core makes PEPA tools available within the Eclipse framework.

Key Plug-ins

- PEPAto provides core services for PEPA. It can be used as a library by third-party applications.
- Eclipse Core makes PEPA tools available within the Eclipse framework.
- Eclipse UI implements a rich graphical user interface including an editor for PEPA descriptions and views for performance evaluation.

Services Available in PEPAto

Services Available in PEPAto

- In-memory model representation either programmatically or through parsing.

Services Available in PEPAto

- In-memory model representation either programmatically or through parsing.
- Static analysis.

Services Available in PEPAto

- In-memory model representation either programmatically or through parsing.
- Static analysis.
- State space derivation.

Services Available in PEPAto

- In-memory model representation either programmatically or through parsing.
- Static analysis.
- State space derivation.
- Calculation of steady-state probability distribution.

Services Available in PEPAto

- In-memory model representation either programmatically or through parsing.
- Static analysis.
- State space derivation.
- Calculation of steady-state probability distribution.
- Throughput and utilisation analysis.

Services Available in PEPAto

- In-memory model representation either programmatically or through parsing.
- Static analysis.
- State space derivation.
- Calculation of steady-state probability distribution.
- Throughput and utilisation analysis.

Static Analysis

Static analysis checks the well-formedness of a model *prior* to inferring the derivation graph of the system.

Static Analysis

Static analysis checks the well-formedness of a model *prior* to inferring the derivation graph of the system.
The output of this tool is a list of messages grouped into two categories:

Static Analysis

Static analysis checks the well-formedness of a model *prior* to inferring the derivation graph of the system.
The output of this tool is a list of messages grouped into two categories:

- Error messages prevent further model analysis (e.g. state space derivation)

Static Analysis

Static analysis checks the well-formedness of a model *prior* to inferring the derivation graph of the system.
The output of this tool is a list of messages grouped into two categories:

- Error messages prevent further model analysis (e.g. state space derivation)
- Warning messages are less severe and allow further processing

Basic Analysis

An Error message is reported for:

Basic Analysis

An Error message is reported for:

- Rate not declared

Basic Analysis

An Error message is reported for:

- Rate not declared
- Process not defined

Basic Analysis

An Error message is reported for:

- Rate not declared
- Process not defined
- Multiple rate definitions

Basic Analysis

An Error message is reported for:

- Rate not declared
- Process not defined
- Multiple rate definitions
- Multiple process definitions

Basic Analysis

An Error message is reported for:

- Rate not declared
- Process not defined
- Multiple rate definitions
- Multiple process definitions

A Warning message is reported for:

Basic Analysis

An Error message is reported for:

- Rate not declared
- Process not defined
- Multiple rate definitions
- Multiple process definitions

A Warning message is reported for:

- Rate not used

Basic Analysis

An Error message is reported for:

- Rate not declared
- Process not defined
- Multiple rate definitions
- Multiple process definitions

A Warning message is reported for:

- Rate not used
- Process definition not used

Local Deadlock

A local deadlock is a condition that may occur in a synchronisation when a shared action can never be performed by one of the involved components.

Local Deadlock

A local deadlock is a condition that may occur in a synchronisation when a shared action can never be performed by one of the involved components.

$$
\begin{aligned}
P1 &\stackrel{def}{=} (\alpha, r).P2 \\
P2 &\stackrel{def}{=} (\gamma, t).P1 \\
Q1 &\stackrel{def}{=} (\beta, s).Q2 \\
Q2 &\stackrel{def}{=} (\epsilon, v).Q1 \\
P1 &\underset{\{\alpha\}}{\bowtie} Q1
\end{aligned}
$$

Complete Action Type Set

The complete action type set $\mathcal{A}(P)$ of a component P is the set of all the action types which may be performed by the component during its evolution.

Complete Action Type Set

The complete action type set $\mathcal{A}(P)$ of a component P is the set of all the action types which may be performed by the component during its evolution.

- Constant $A \stackrel{def}{=} P$:
  $\mathcal{A}(A) = \mathcal{A}(P)$

Complete Action Type Set

The complete action type set $\mathcal{A}(P)$ of a component P is the set of all the action types which may be performed by the component during its evolution.

- Constant $A \stackrel{def}{=} P$:
  $\mathcal{A}(A) = \mathcal{A}(P)$
- Choice $P + Q$:
  $\mathcal{A}(P + Q) = \mathcal{A}(P) \cup \mathcal{A}(Q)$

Complete Action Type Set

The complete action type set $\mathcal{A}(P)$ of a component P is the set of all the action types which may be performed by the component during its evolution.

- Constant $A \stackrel{def}{=} P$:
  $\mathcal{A}(A) = \mathcal{A}(P)$
- Choice $P + Q$:
  $\mathcal{A}(P + Q) = \mathcal{A}(P) \cup \mathcal{A}(Q)$
- Cooperation $P \underset{L}{\bowtie} Q$:
  $\mathcal{A}(P \underset{L}{\bowtie} Q) = \mathcal{A}(P) \cup \mathcal{A}(Q)$

Complete Action Type Set

The complete action type set $\mathcal{A}(P)$ of a component P is the set of all the action types which may be performed by the component during its evolution.

- Constant $A \stackrel{def}{=} P$:
  $\mathcal{A}(A) = \mathcal{A}(P)$
- Choice $P + Q$:
  $\mathcal{A}(P + Q) = \mathcal{A}(P) \cup \mathcal{A}(Q)$
- Cooperation $P \underset{L}{\bowtie} Q$:
  $\mathcal{A}(P \underset{L}{\bowtie} Q) = \mathcal{A}(P) \cup \mathcal{A}(Q)$
- Prefix $(\alpha, r).P$:
  $\mathcal{A}\big((\alpha, r).P\big) = \{\alpha\} \cup \mathcal{A}(P)$

Complete Action Type Set

The complete action type set $\mathcal{A}(P)$ of a component P is the set of all the action types which may be performed by the component during its evolution.

- Constant $A \stackrel{def}{=} P$:
  $\mathcal{A}(A) = \mathcal{A}(P)$
- Choice $P + Q$:
  $\mathcal{A}(P + Q) = \mathcal{A}(P) \cup \mathcal{A}(Q)$
- Cooperation $P \underset{L}{\bowtie} Q$:
  $\mathcal{A}(P \underset{L}{\bowtie} Q) = \mathcal{A}(P) \cup \mathcal{A}(Q)$
- Prefix $(\alpha, r).P$:
  $\mathcal{A}\big((\alpha, r).P\big) = \{\alpha\} \cup \mathcal{A}(P)$
- Hiding $P\backslash\{L\}$:
  $\mathcal{A}\big(P\backslash\{L\}\big) = \mathcal{A}(P) - L$

Example

$$
\begin{array}{lcl}
P1 & \stackrel{def}{=} & (\alpha, r).P2 \\
P2 & \stackrel{def}{=} & (\gamma, t).P1 \\
Q1 & \stackrel{def}{=} & (\beta, s).Q2 \\
Q2 & \stackrel{def}{=} & (\epsilon, v).Q1 \\
P1 & \underset{\{\alpha\}}{\bowtie} & Q1
\end{array}
$$

Example

$$
\begin{array}{lcl}
P1 & \stackrel{def}{=} & (\alpha, r).P2 \\
P2 & \stackrel{def}{=} & (\gamma, t).P1 \\
Q1 & \stackrel{def}{=} & (\beta, s).Q2 \\
Q2 & \stackrel{def}{=} & (\epsilon, v).Q1 \\
P1 & \underset{\{\alpha\}}{\bowtie} & Q1
\end{array}
$$

$$\mathcal{A}(P1) = \{\alpha, \gamma\}$$

Example

$$
\begin{array}{lcl}
P1 & \stackrel{def}{=} & (\alpha, r).P2 \\
P2 & \stackrel{def}{=} & (\gamma, t).P1 \\
Q1 & \stackrel{def}{=} & (\beta, s).Q2 \\
Q2 & \stackrel{def}{=} & (\epsilon, v).Q1 \\
P1 & \underset{\{\alpha\}}{\bowtie} & Q1
\end{array}
$$

$$\mathcal{A}(P1) = \{\alpha, \gamma\}$$
$$\mathcal{A}(P2) = \{\alpha, \gamma\}$$

Example

$$
\begin{array}{lcl}
P1 & \stackrel{def}{=} & (\alpha, r).P2 \\
P2 & \stackrel{def}{=} & (\gamma, t).P1 \\
Q1 & \stackrel{def}{=} & (\beta, s).Q2 \\
Q2 & \stackrel{def}{=} & (\epsilon, v).Q1 \\
P1 & \underset{\{\alpha\}}{\bowtie} & Q1
\end{array}
$$

$$
\begin{aligned}
\mathcal{A}(P1) &= \{\alpha, \gamma\} \\
\mathcal{A}(P2) &= \{\alpha, \gamma\} \\
\mathcal{A}(Q1) &= \{\beta, \epsilon\}
\end{aligned}
$$

Example

$$
\begin{array}{lcl}
P1 & \stackrel{def}{=} & (\alpha, r).P2 \\
P2 & \stackrel{def}{=} & (\gamma, t).P1 \\
Q1 & \stackrel{def}{=} & (\beta, s).Q2 \\
Q2 & \stackrel{def}{=} & (\epsilon, v).Q1 \\
P1 & \underset{\{\alpha\}}{\bowtie} & Q1
\end{array}
$$

$$
\begin{aligned}
\mathcal{A}(P1) &= \{\alpha, \gamma\} \\
\mathcal{A}(P2) &= \{\alpha, \gamma\} \\
\mathcal{A}(Q1) &= \{\beta, \epsilon\} \\
\mathcal{A}(Q2) &= \{\beta, \epsilon\}
\end{aligned}
$$

Example

$$
\begin{array}{lcl}
P1 & \stackrel{def}{=} & (\alpha, r).P2 \\
P2 & \stackrel{def}{=} & (\gamma, t).P1 \\
Q1 & \stackrel{def}{=} & (\beta, s).Q2 \\
Q2 & \stackrel{def}{=} & (\epsilon, v).Q1 \\
P1 & \underset{\{\alpha\}}{\bowtie} & Q1
\end{array}
$$

$$
\begin{aligned}
\mathcal{A}(P1) &= \{\alpha, \gamma\} \\
\mathcal{A}(P2) &= \{\alpha, \gamma\} \\
\mathcal{A}(Q1) &= \{\beta, \epsilon\} \\
\mathcal{A}(Q2) &= \{\beta, \epsilon\}
\end{aligned}
$$

Local Deadlock Detection

A cooperation $P \underset{L}{\bowtie} Q$ gives rise to a local deadlock if

$$\exists\, \alpha \in L : \alpha \notin \mathcal{A}(P) \cap \mathcal{A}(Q), \alpha \in \mathcal{A}(P) \cup \mathcal{A}(Q)$$

Redundant Action

A redundant action is an action type specified in a cooperation set which cannot be carried out by either of the components involved.

Redundant Action

A redundant action is an action type specified in a cooperation set which cannot be carried out by either of the components involved.

$$
\begin{array}{lcl}
P1 & \stackrel{def}{=} & (\alpha, r).P2 \\
P2 & \stackrel{def}{=} & (\gamma, t).P1 \\
Q1 & \stackrel{def}{=} & (\beta, s).Q2 \\
Q2 & \stackrel{def}{=} & (\epsilon, v).Q1 \\
P1 & \underset{\{\alpha,\delta\}}{\bowtie} & Q1
\end{array}
$$

Redundant Action

A redundant action is an action type specified in a cooperation set which cannot be carried out by either of the components involved.

$$
\begin{array}{lcl}
P1 & \stackrel{def}{=} & (\alpha, r).P2 \\
P2 & \stackrel{def}{=} & (\gamma, t).P1 \\
Q1 & \stackrel{def}{=} & (\beta, s).Q2 \\
Q2 & \stackrel{def}{=} & (\epsilon, v).Q1 \\
P1 & \underset{\{\alpha,\delta\}}{\bowtie} & Q1
\end{array}
$$

Redundant Action

A redundant action is an action type specified in a cooperation set which cannot be carried out by either of the components involved.

$$
\begin{array}{lcl}
P1 & \stackrel{def}{=} & (\alpha, r).P2 \\
P2 & \stackrel{def}{=} & (\gamma, t).P1 \\
Q1 & \stackrel{def}{=} & (\beta, s).Q2 \\
Q2 & \stackrel{def}{=} & (\epsilon, v).Q1 \\
P1 & \underset{\{\alpha,\delta\}}{\bowtie} & Q1
\end{array}
$$

Redundant Action Detection

A cooperation $P \underset{L}{\bowtie} Q$ has a redundant action type α if

$$\exists\, \alpha \in L : \alpha \notin \mathcal{A}(P) \cup \mathcal{A}(Q)$$

Non-guarded Recursive Definition

Consider the following model snippet:

$$
\begin{array}{lcl}
\ldots & & \\
P1 & \stackrel{def}{=} & P2 \parallel P3 \\
P2 & \stackrel{def}{=} & (\gamma, t).P1 \\
\ldots & &
\end{array}
$$

Non-guarded Recursive Definition

Consider the following model snippet:

$$\begin{array}{lcl} \ldots & & \\ P1 & \stackrel{def}{=} & P2 \parallel P3 \\ P2 & \stackrel{def}{=} & (\gamma, t).P1 \\ \ldots & & \end{array}$$

The derivation graph of component $P1$ gives rise to infinite-depth left recursion:

$$\begin{array}{lcl} P1 & \stackrel{(\gamma,t)}{\longrightarrow} & P2 \parallel P3 \parallel P3 \\ & \stackrel{(\gamma,t)}{\longrightarrow} & P2 \parallel P3 \parallel P3 \parallel P3 \\ & \stackrel{(\gamma,t)}{\longrightarrow} & P2 \parallel P3 \parallel P3 \parallel P3 \parallel P3 \\ & \stackrel{(\gamma,t)}{\longrightarrow} & \ldots \end{array}$$

Used Definition Set

The used definition set $\mathcal{U}(P)$ of a component P is the set of all the constants used by P during its evolution.

It is computed as follows:

Used Definition Set

The used definition set $\mathcal{U}(P)$ of a component P is the set of all the constants used by P during its evolution.
It is computed as follows:

- Constant $A \stackrel{def}{=} P$:
  $\mathcal{U}(A) = \{A\} \cup \mathcal{U}(P)$

Used Definition Set

The used definition set $\mathcal{U}(P)$ of a component P is the set of all the constants used by P during its evolution.
It is computed as follows:

- Constant $A \stackrel{def}{=} P$:
  $\mathcal{U}(A) = \{A\} \cup \mathcal{U}(P)$
- Choice $P + Q$:
  $\mathcal{U}(P + Q) = \mathcal{U}(P) \cup \mathcal{U}(Q)$

Used Definition Set

The used definition set $\mathcal{U}(P)$ of a component P is the set of all the constants used by P during its evolution.
It is computed as follows:

- Constant $A \stackrel{def}{=} P$:
  $\mathcal{U}(A) = \{A\} \cup \mathcal{U}(P)$
- Choice $P + Q$:
  $\mathcal{U}(P + Q) = \mathcal{U}(P) \cup \mathcal{U}(Q)$
- Cooperation $P \underset{L}{\bowtie} Q$:
  $\mathcal{U}(P \underset{L}{\bowtie} Q) = \mathcal{U}(P) \cup \mathcal{U}(Q)$

Used Definition Set

The used definition set $\mathcal{U}(P)$ of a component P is the set of all the constants used by P during its evolution.
It is computed as follows:

- Constant $A \stackrel{def}{=} P$:
  $\mathcal{U}(A) = \{A\} \cup \mathcal{U}(P)$
- Choice $P + Q$:
  $\mathcal{U}(P + Q) = \mathcal{U}(P) \cup \mathcal{U}(Q)$
- Cooperation $P \underset{L}{\bowtie} Q$:
  $\mathcal{U}(P \underset{L}{\bowtie} Q) = \mathcal{U}(P) \cup \mathcal{U}(Q)$
- Prefix $(\alpha, r).P$:
  $\mathcal{U}\big((\alpha, r).P\big) = \mathcal{U}(P)$

Used Definition Set

The used definition set $\mathcal{U}(P)$ of a component P is the set of all the constants used by P during its evolution.
It is computed as follows:

- Constant $A \stackrel{def}{=} P$:
  $\mathcal{U}(A) = \{A\} \cup \mathcal{U}(P)$
- Choice $P + Q$:
  $\mathcal{U}(P + Q) = \mathcal{U}(P) \cup \mathcal{U}(Q)$
- Cooperation $P \underset{L}{\bowtie} Q$:
  $\mathcal{U}(P \underset{L}{\bowtie} Q) = \mathcal{U}(P) \cup \mathcal{U}(Q)$
- Prefix $(\alpha, r).P$:
  $\mathcal{U}\big((\alpha, r).P\big) = \mathcal{U}(P)$
- Hiding $P\backslash\{L\}$:
  $\mathcal{U}\big(P\backslash\{L\}\big) = \mathcal{U}(P)$

Example

$$
\begin{array}{lcl}
\ldots & & \\
P1 & \stackrel{def}{=} & P2 \parallel P3 \\
P2 & \stackrel{def}{=} & (\gamma, t).P1 \\
\ldots & &
\end{array}
$$

Example

$$
\begin{array}{lcl}
\ldots & & \\
P1 & \stackrel{def}{=} & P2 \parallel P3 \\
P2 & \stackrel{def}{=} & (\gamma, t).P1 \\
\ldots & &
\end{array}
$$

Non-guarded Definition Detection

For each process definition $A \stackrel{def}{=} P$ compute $\mathcal{U}(P)$.
A has infinite recursion if it defines a cooperation and $A \in \mathcal{U}(P)$

Example

$$
\begin{array}{lcl}
\ldots & & \\
P1 & \stackrel{def}{=} & P2 \parallel P3 \\
P2 & \stackrel{def}{=} & (\gamma, t).P1 \\
\ldots & &
\end{array}
$$

Non-guarded Definition Detection

For each process definition $A \stackrel{def}{=} P$ compute $\mathcal{U}(P)$.
A has infinite recursion if it defines a cooperation and $A \in \mathcal{U}(P)$

$$\mathcal{U}(P2 \parallel P3) = \{P1, P2, P3, \ldots\}$$

Example

$$
\begin{array}{lcl}
\ldots & & \\
P1 & \stackrel{def}{=} & P2 \parallel P3 \\
P2 & \stackrel{def}{=} & (\gamma, t).P1 \\
\ldots & &
\end{array}
$$

Non-guarded Definition Detection

For each process definition $A \stackrel{def}{=} P$ compute $\mathcal{U}(P)$.
A has infinite recursion if it defines a cooperation and $A \in \mathcal{U}(P)$

$$\mathcal{U}\big(P2 \parallel P3\big) = \{P1, P2, P3, \ldots\}$$
$$\mathcal{U}\big((\gamma, t).P1\big) = \{P1, P2, P3, \ldots\}$$

Example

$$
\begin{array}{lcl}
\ldots & & \\
P1 & \stackrel{def}{=} & P2 \parallel P3 \\
P2 & \stackrel{def}{=} & (\gamma, t).P1 \\
\ldots & &
\end{array}
$$

Non-guarded Definition Detection

For each process definition $A \stackrel{def}{=} P$ compute $\mathcal{U}(P)$.
A has infinite recursion if it defines a cooperation and $A \in \mathcal{U}(P)$

$$\mathcal{U}(P2 \parallel P3) = \{P1, P2, P3, \ldots\}$$
$$\mathcal{U}((\gamma, t).P1) = \{P1, P2, P3, \ldots\}$$

Web Service Composition: Introduction

We consider an example of a business application which is composed from a number of offered web services.

Web Service Composition: Introduction

We consider an example of a business application which is composed from a number of offered web services.

A user accesses the application via an SMS message requesting directions to the nearest facility (post-office, restaurant, bank etc.) and receives a response as an MMS message containing a map.

Web Service Composition: Introduction

We consider an example of a business application which is composed from a number of offered web services.

A user accesses the application via an SMS message requesting directions to the nearest facility (post-office, restaurant, bank etc.) and receives a response as an MMS message containing a map.

Since the application involves a users' current location there is an access control issue since it must be ensured that the web service consumer has the requisite authority to execute the web service it requests.

Web Service Composition: Introduction

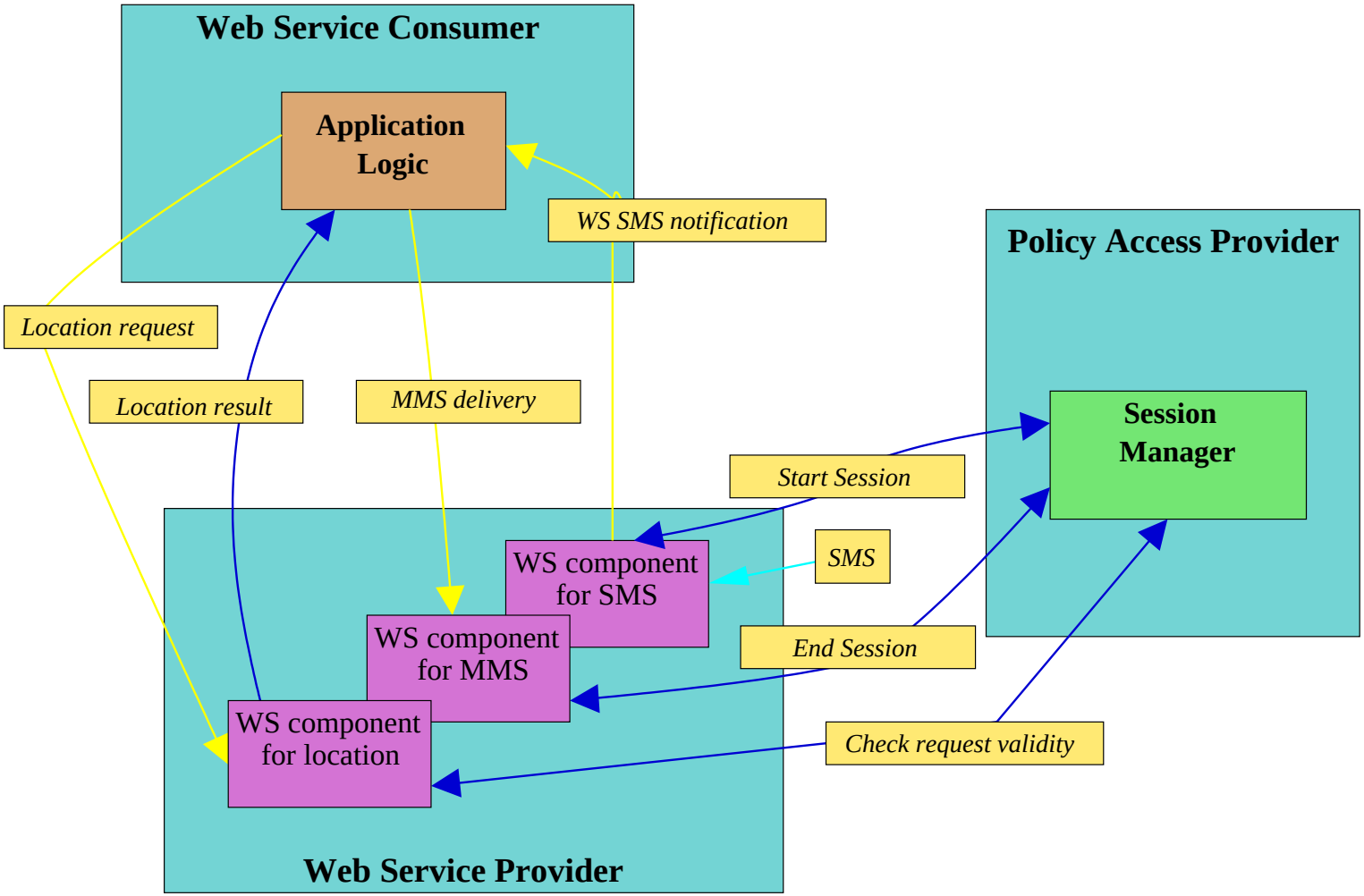
We consider an example of a business application which is composed from a number of offered web services.

A user accesses the application via an SMS message requesting directions to the nearest facility (post-office, restaurant, bank etc.) and receives a response as an MMS message containing a map.

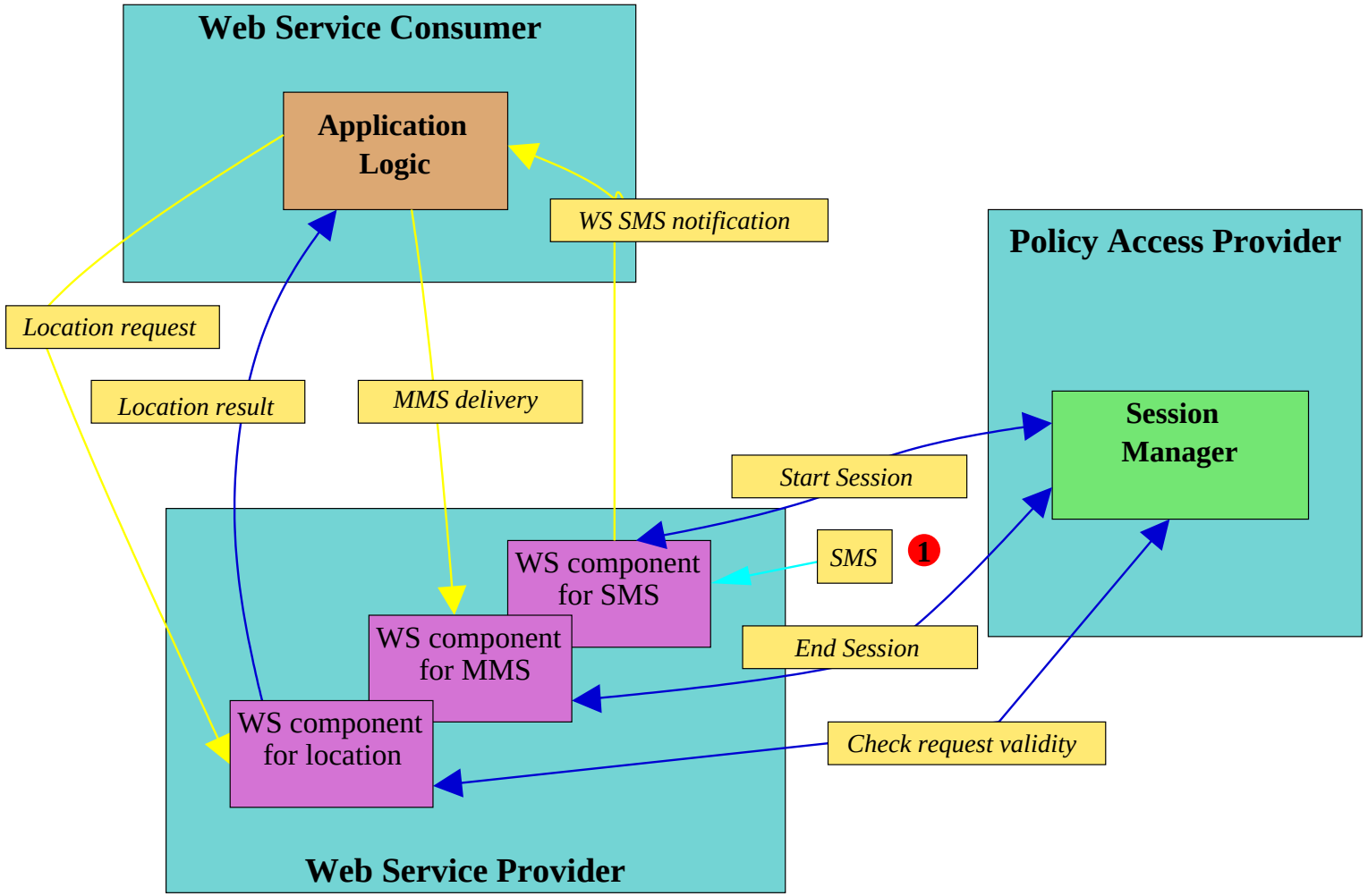
Since the application involves a users' current location there is an access control issue since it must be ensured that the web service consumer has the requisite authority to execute the web service it requests.

Moreover the service provider imposes a restriction that only one request may be handled for each SMS message received.

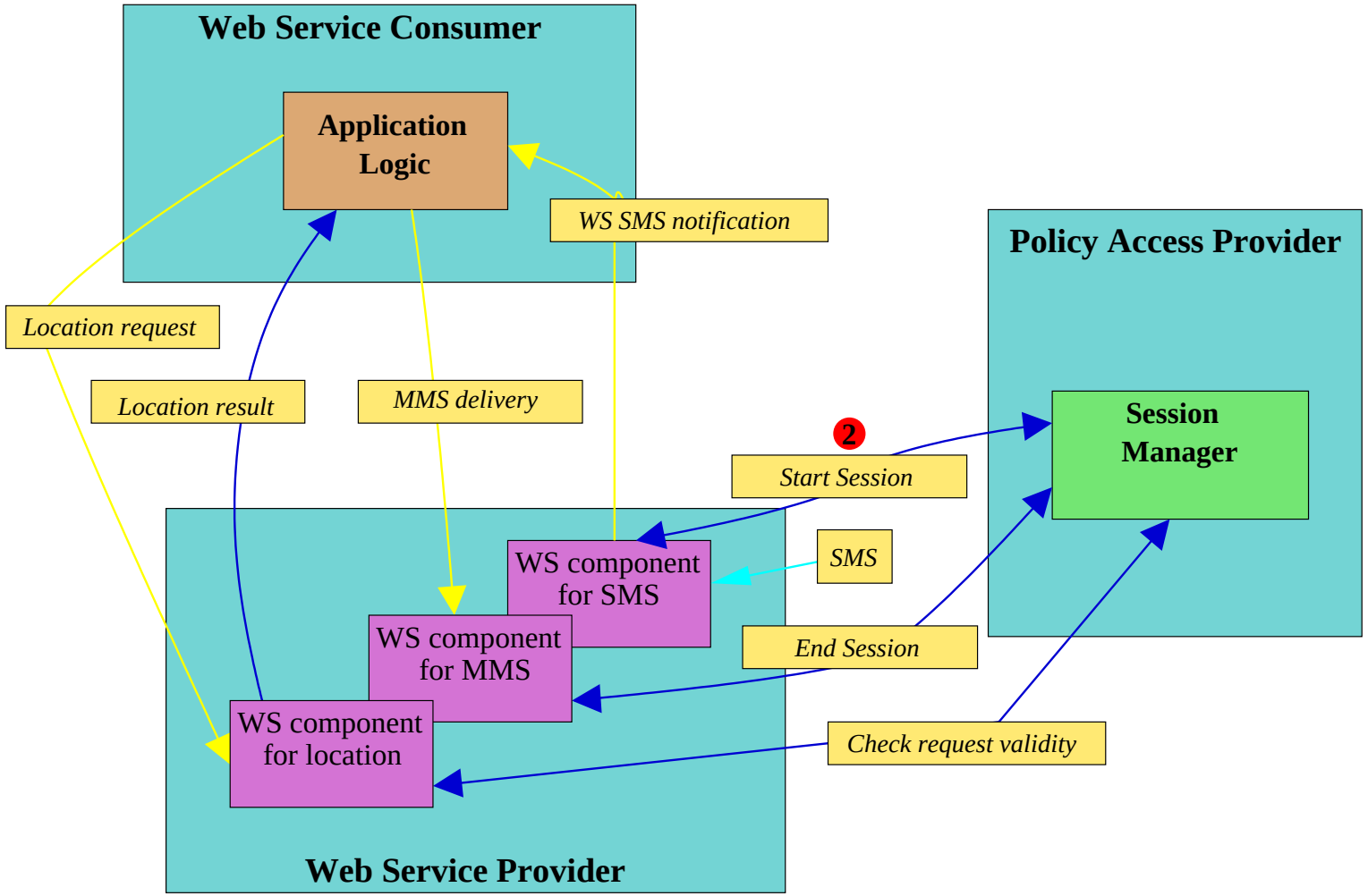
Schematic view



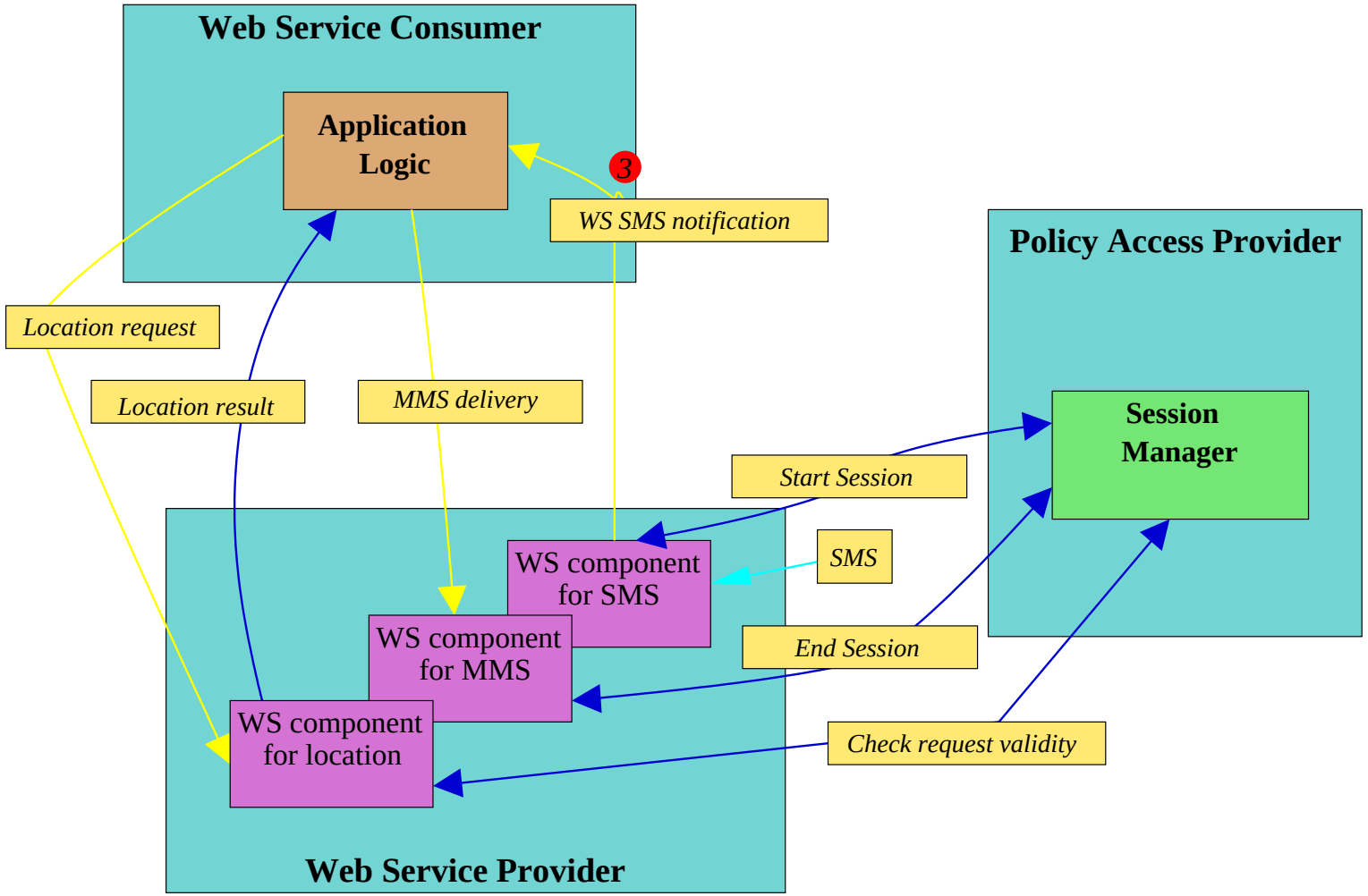
Schematic view



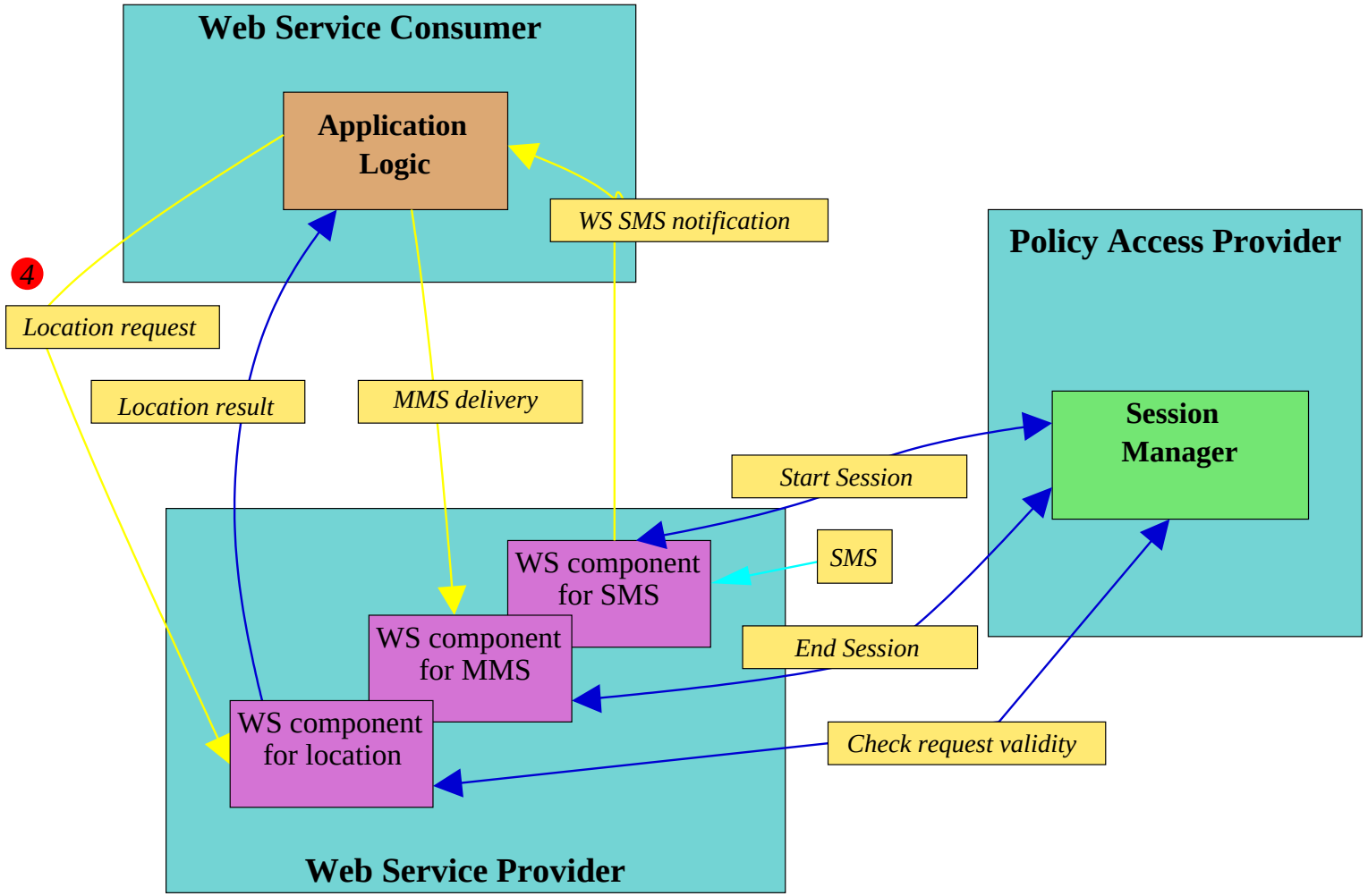
Schematic view



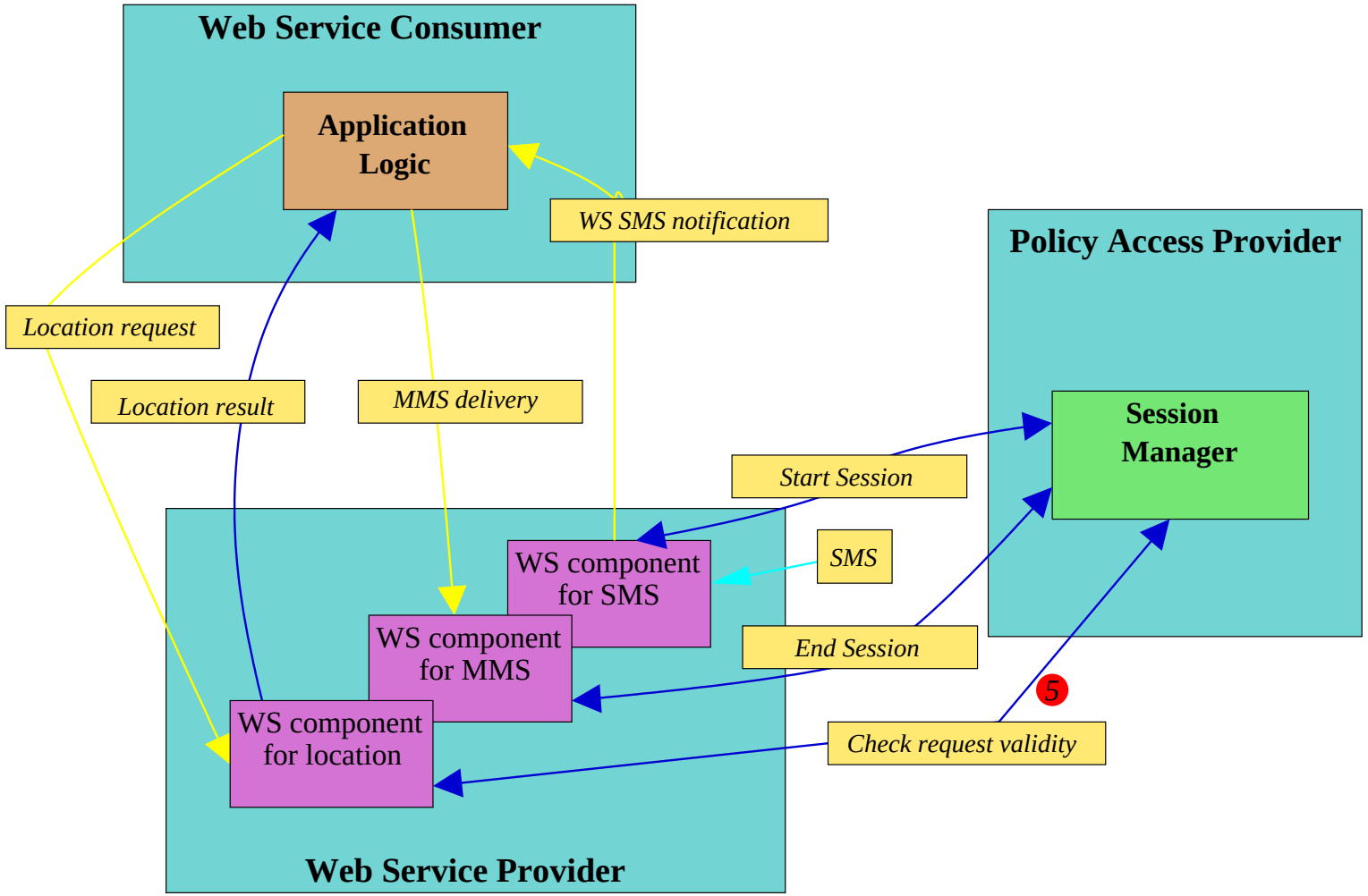
Schematic view



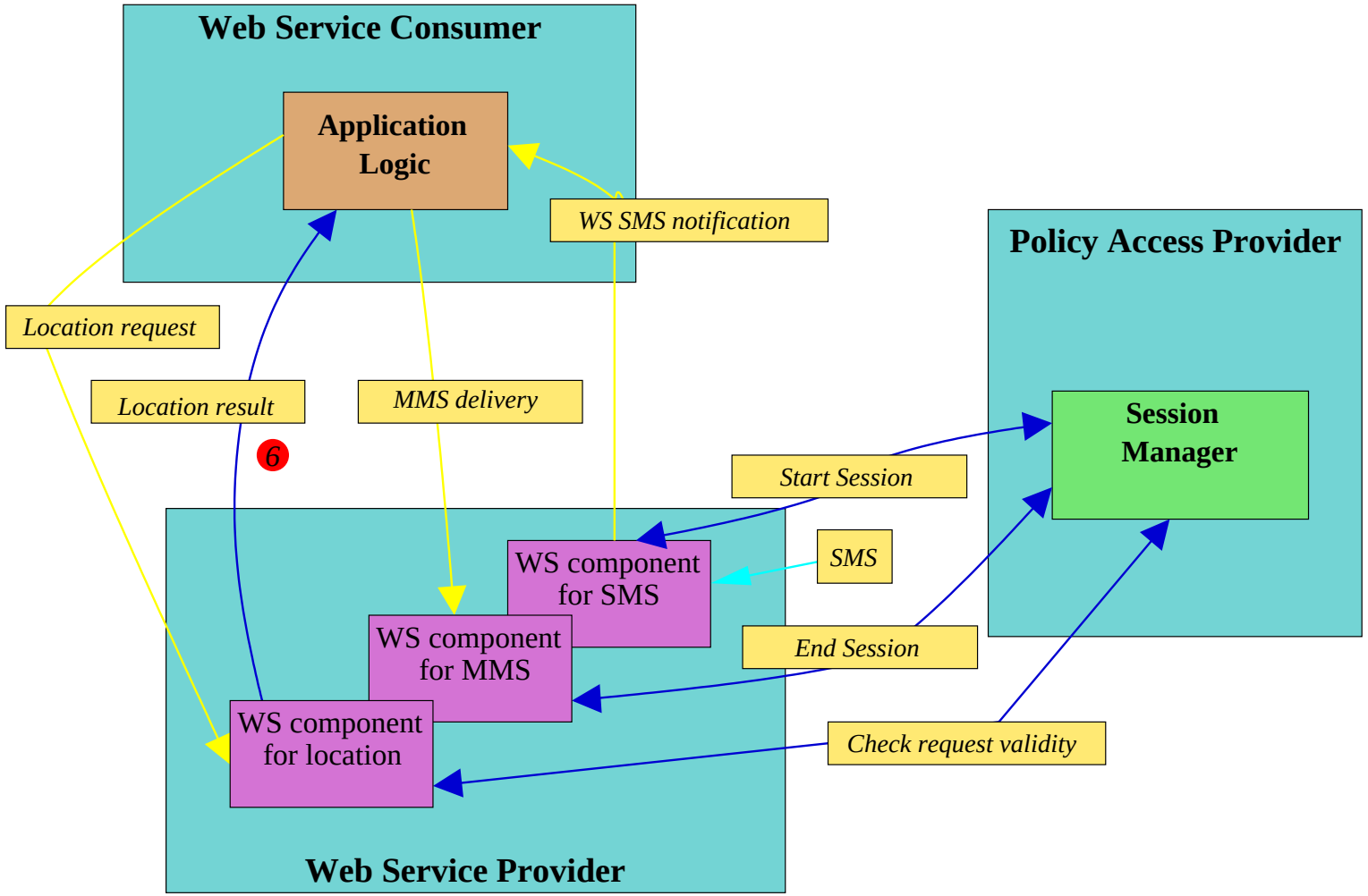
Schematic view



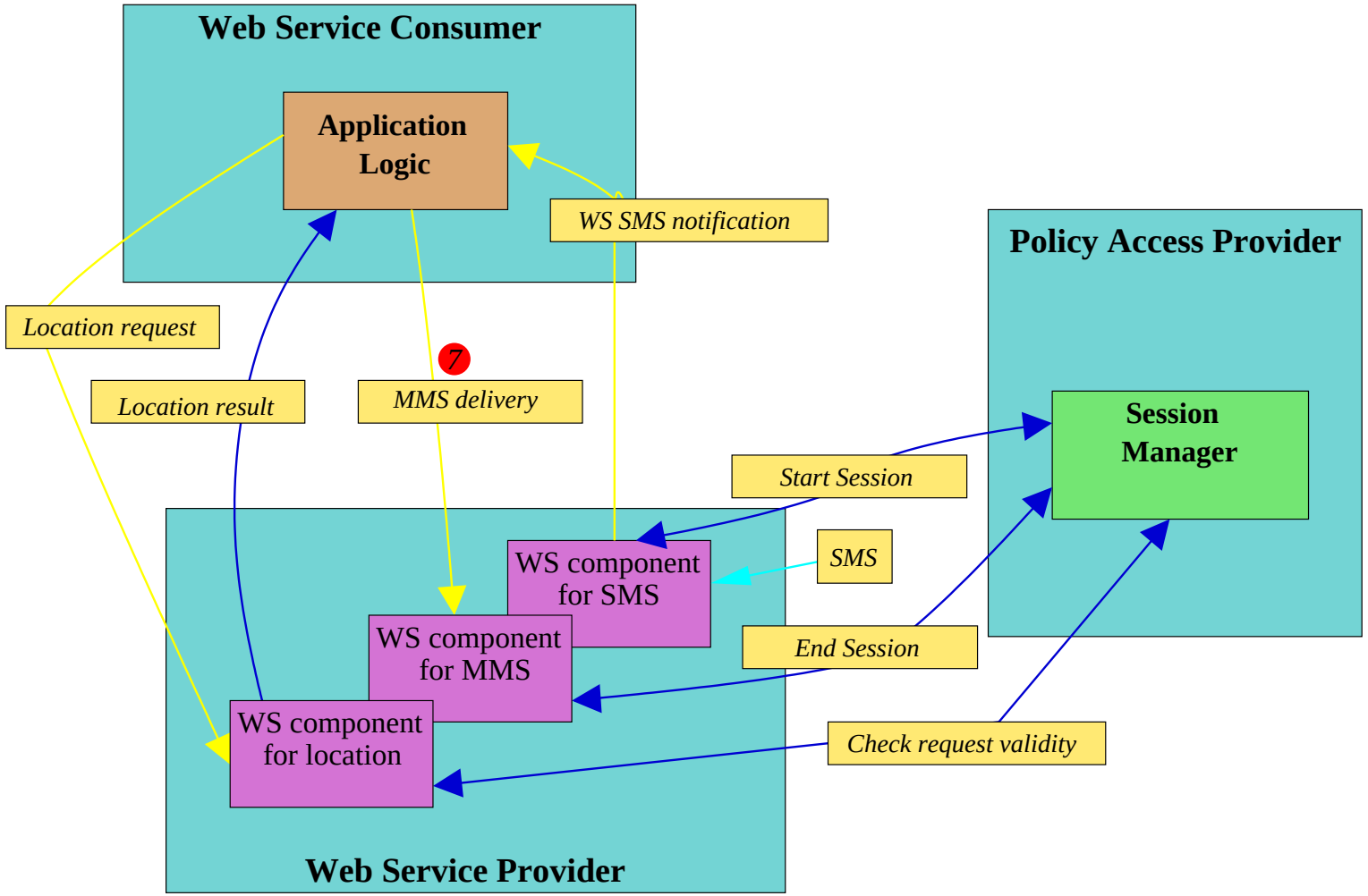
Schematic view



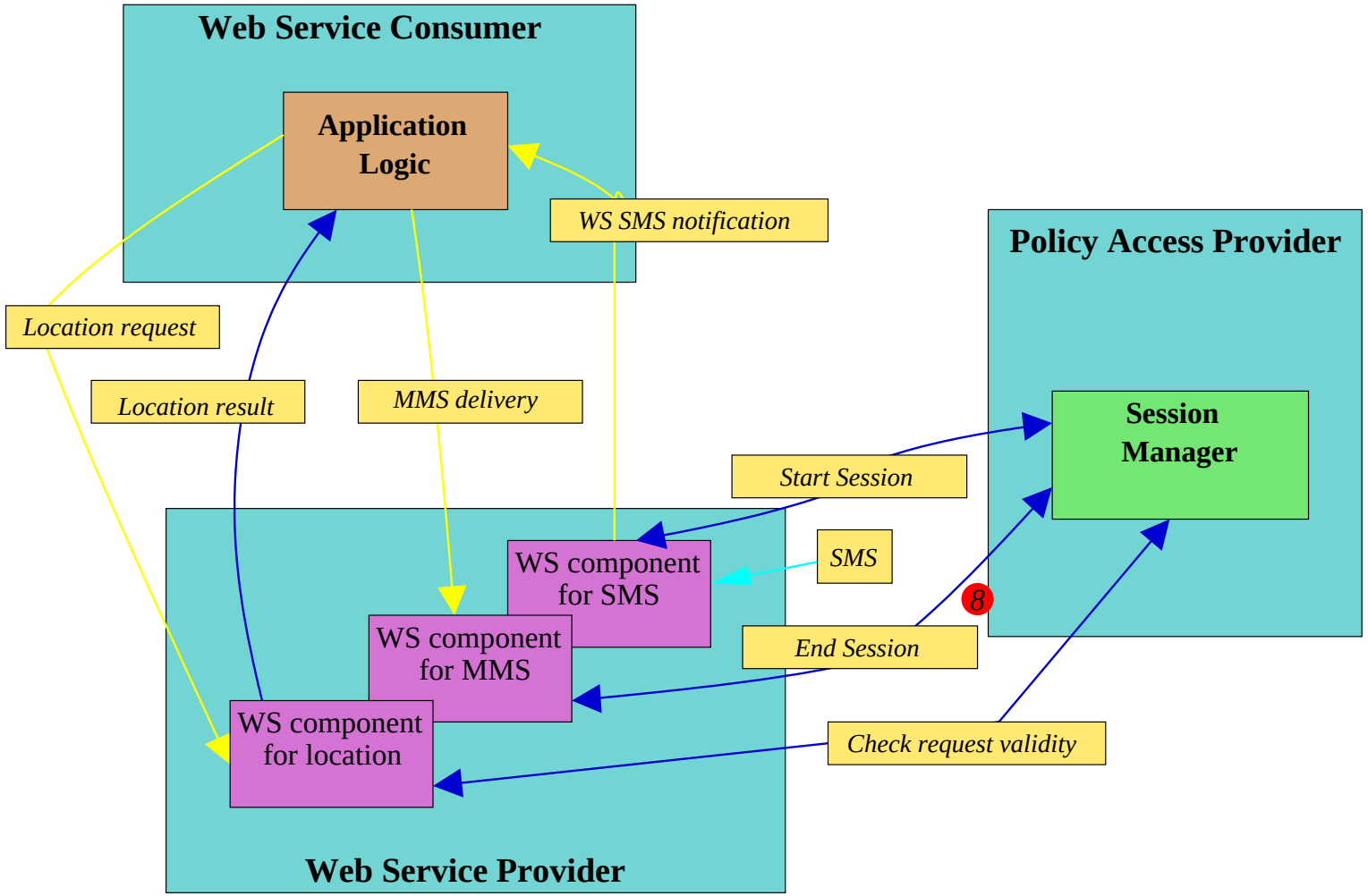
Schematic view



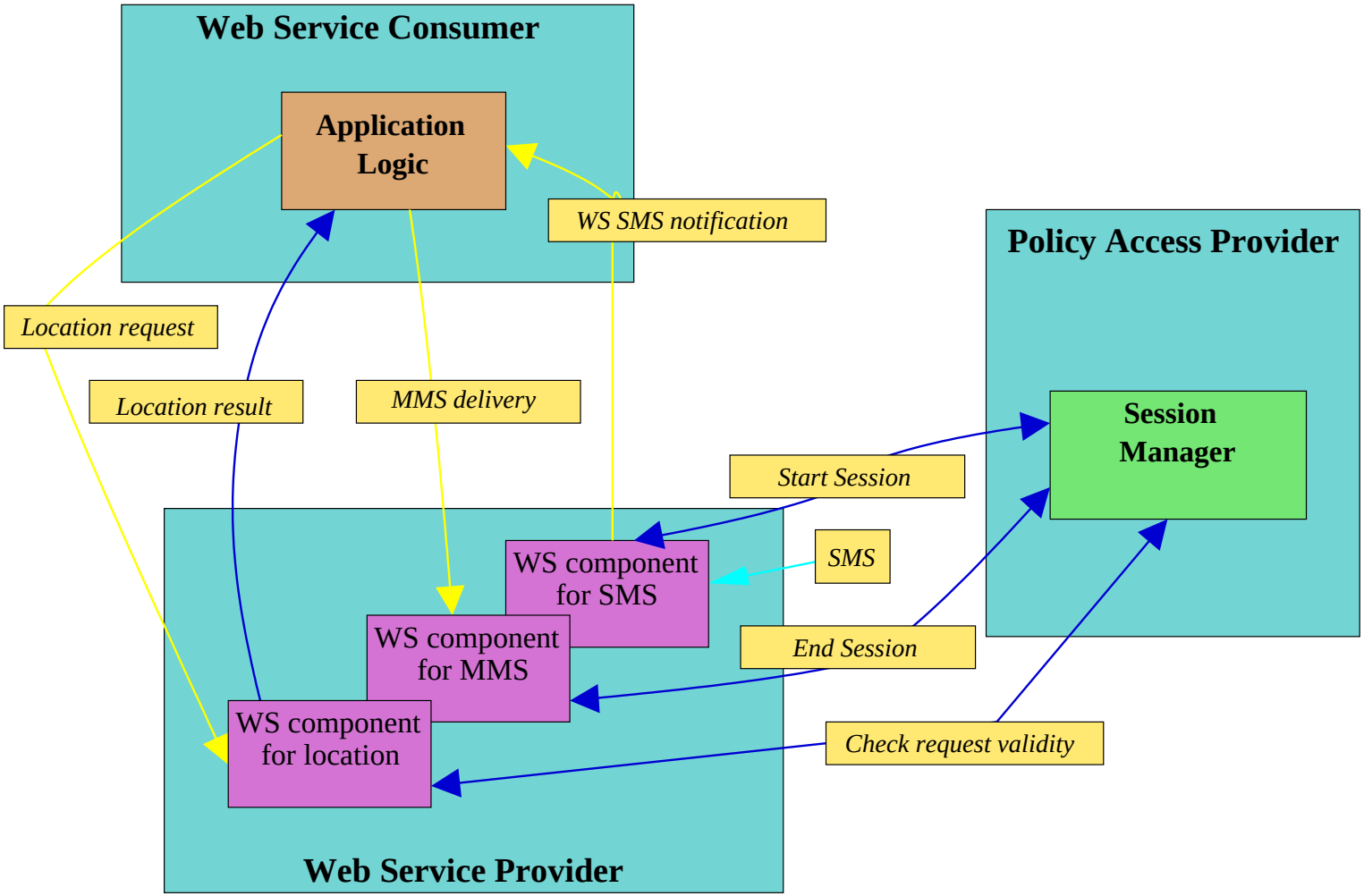
Schematic view



Schematic view



Schematic view



The PEPA model

The PEPA model of the system consists of four components:

The PEPA model

The PEPA model of the system consists of four components:

- The user;
- The web service provider;
- The web service consumer, and
- The policy access provider.

The PEPA model

The PEPA model of the system consists of four components:

- The user;
- The web service provider;
- The web service consumer, and
- The policy access provider.

The Web Service Provider consists of three distinct elements but the web service consumer is associated with a session which accesses each element in sequence.

The PEPA model

The PEPA model of the system consists of four components:

- The user;
- The web service provider;
- The web service consumer, and
- The policy access provider.

The Web Service Provider consists of three distinct elements but the web service consumer is associated with a session which accesses each element in sequence.

Concurrency is introduced into the model by allowing multiple sessions rather than by representing the constituent web services separately.

Component *Customer*

The customer's behaviour is simply modelled with two local states.

Component *Customer*

The customer's behaviour is simply modelled with two local states.

$$
\begin{aligned}
Customer &\stackrel{def}{=} (getSMS, r_1).Customer_1 \\
Customer_1 &\stackrel{def}{=} (getMap, \top).Customer \\
&\quad + \quad (get404, \top).Customer
\end{aligned}
$$

Component *Customer*

The customer's behaviour is simply modelled with two local states.

$$\begin{aligned} Customer &\stackrel{def}{=} (getSMS, r_1).Customer_1 \\ Customer_1 &\stackrel{def}{=} (getMap, \top).Customer \\ &+ (get404, \top).Customer \end{aligned}$$

We associate the user-perceived system performance with the throughput of the *getMap* action which can be calculated directly from the steady state probability distribution of the underlying Markov chain.

Component *WSConsumer*

Once a session has been started, it initiates a request for the user's current location and waits for a response.

Component *WSConsumer*

Once a session has been started, it initiates a request for the user's current location and waits for a response.
For valid requests, location is returned and used to compute the appropriate map, which is then sent via an MMS message, using the web service.

Component *WSConsumer*

Once a session has been started, it initiates a request for the user's current location and waits for a response.
For valid requests, location is returned and used to compute the appropriate map, which is then sent via an MMS message, using the web service.

$$
\begin{aligned}
WSConsumer &\stackrel{def}{=} (notify, \top).WSConsumer_2 \\
WSConsumer_2 &\stackrel{def}{=} (locReq, r_4).WSConsumer_3 \\
WSConsumer_3 &\stackrel{def}{=} (locRes, \top).WSConsumer_4 \\
&\quad + \quad (locErr, \top).WSConsumer \\
WSConsumer_4 &\stackrel{def}{=} (compute, r_7).WSConsumer_5 \\
WSConsumer_5 &\stackrel{def}{=} (sendMMS, r_9).WSConsumer
\end{aligned}
$$

Component *WSProvider*

The use of sessions restricts a user's access to the services of the Web Service Provider to be sequential.

Component *WSProvider*

The use of sessions restricts a user's access to the services of the Web Service Provider to be sequential.

We assume that there is a distinct instance of the component *WSProvider* for each distinct session.

Component *WSProvider*

The use of sessions restricts a user's access to the services of the Web Service Provider to be sequential.

We assume that there is a distinct instance of the component *WSProvider* for each distinct session.

The *checkValid* action is represented twice, to capture the two possible distinct outcomes of the action.

Component *WSProvider*

The use of sessions restricts a user's access to the services of the Web Service Provider to be sequential.

We assume that there is a distinct instance of the component *WSProvider* for each distinct session.

The *checkValid* action is represented twice, to capture the two possible distinct outcomes of the action.

- If the check is successful the location must be returned to the Web Service Consumer in the form of a map (*getMap*).

Component *WSProvider*

The use of sessions restricts a user's access to the services of the Web Service Provider to be sequential.

We assume that there is a distinct instance of the component *WSProvider* for each distinct session.

The *checkValid* action is represented twice, to capture the two possible distinct outcomes of the action.

- If the check is successful the location must be returned to the Web Service Consumer in the form of a map (*getMap*).
- If the check revealed an invalid request (*locErr*) then an error must be returned to the Web Service Consumer (*get404*) and the session terminated (*stopSession*).

Component *WSProvider*

$$
\begin{aligned}
WSProvider &\stackrel{def}{=} (getSMS, \top).WSProvider_2 \\
WSProvider_2 &\stackrel{def}{=} (startSession, r_2).WSProvider_3 \\
WSProvider_3 &\stackrel{def}{=} (notify, r_3).WSProvider_4 \\
WSProvider_4 &\stackrel{def}{=} (locReq, \top).WSProvider_5 \\
WSProvider_5 &\stackrel{def}{=} (checkValid, 99 \cdot \top).WSProvider_6 \\
&\quad + \quad (checkValid, \top).WSProvider_{10}
\end{aligned}
$$

Component *WSProvider* cont.

$$
\begin{aligned}
WSProvider_6 &\stackrel{def}{=} (locRes, r_6).WSProvider_7 \\
WSProvider_7 &\stackrel{def}{=} (sendMMS, \top).WSProvider_8 \\
WSProvider_8 &\stackrel{def}{=} (getMap, r_8).WSProvider_9 \\
WSProvider_9 &\stackrel{def}{=} (stopSession, r_2).WSProvider \\
WSProvider_{10} &\stackrel{def}{=} (locErr, r_6).WSProvider_{11} \\
WSProvider_{11} &\stackrel{def}{=} (get404, r_8).WSProvider_9
\end{aligned}
$$

Component *PAProvider*

We consider a stateless implementation of the policy access provider.

$$\begin{aligned} PAProvider &\stackrel{def}{=} (startSession, \top).PAProvider \\ &+ (checkValid, r_5).PAProvider \\ &+ (stopSession, \top).PAProvider \end{aligned}$$

Model Component *WSComp*

The complete system is composed of some number of instances of the components interacting on their shared activities:

$$\begin{aligned} WSComp \stackrel{def}{=} & \big((Customer[N_C] \underset{L_1}{\bowtie} WSProvider[N_{WSP}]) \\ & \underset{L_2}{\bowtie} WSConsumer[N_{WSC}]\big) \\ & \underset{L_3}{\bowtie} PAProvider[N_{PAP}] \end{aligned}$$

where the cooperation sets are

$$\begin{aligned} L_1 &= \{getSMS, getMap, get404\} \\ L_2 &= \{notify, locReq, locRes, locErr, sendMMS\} \\ L_3 &= \{startSession, checkValid, stopSession\} \end{aligned}$$

Parameter Values

param	*value*	*explanation*
r_1	0.0010	rate customers request maps
r_2	0.5	rate session can be started
r_3	0.1	notification exchange between consumer and provider
r_4	0.1	rate requests for location can be satisfied
r_5	0.05	rate the provider can check the validity of the request
r_6	0.1	rate location information can be returned to consumer
r_7	0.05	rate maps can be generated
r_8	0.02	rate MMS messages can be sent from provider to customer
r_9	$10.0 * r_8$	rate MMS messages can be sent via the Web Service

Steady State Analysis for System Tuning

- Suppose that we want to design the system in such a way that it can handle 30 independent customers.

Steady State Analysis for System Tuning

- Suppose that we want to design the system in such a way that it can handle 30 independent customers.
- Some parameters such as the network delays may be constrained by the available technology.

Steady State Analysis for System Tuning

- Suppose that we want to design the system in such a way that it can handle 30 independent customers.
- Some parameters such as the network delays may be constrained by the available technology.
- However, there are a number of degrees of freedom which let her vary, for example, the number of threads of control of the components of the system.

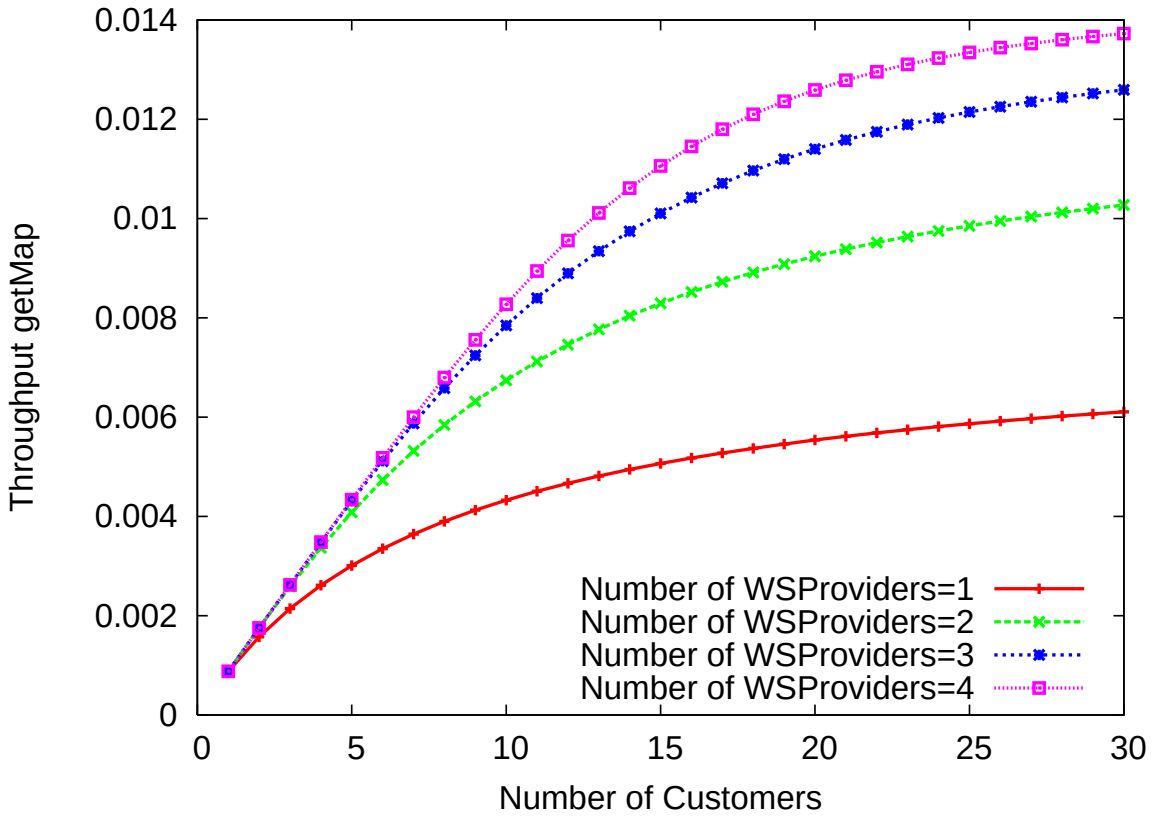
Steady State Analysis for System Tuning

- Suppose that we want to design the system in such a way that it can handle 30 independent customers.
- Some parameters such as the network delays may be constrained by the available technology.
- However, there are a number of degrees of freedom which let her vary, for example, the number of threads of control of the components of the system.
- The aim of the analysis is to deliver a satisfactory service in a cost-effective way.

Steady State Analysis for System Tuning

- Suppose that we want to design the system in such a way that it can handle 30 independent customers.
- Some parameters such as the network delays may be constrained by the available technology.
- However, there are a number of degrees of freedom which let her vary, for example, the number of threads of control of the components of the system.
- The aim of the analysis is to deliver a satisfactory service in a cost-effective way.
- The simplest example of a cost function may be a linearly dependency on the number of copies of a component or the rate at which an activity is performed.

Throughput of the *getMap* action

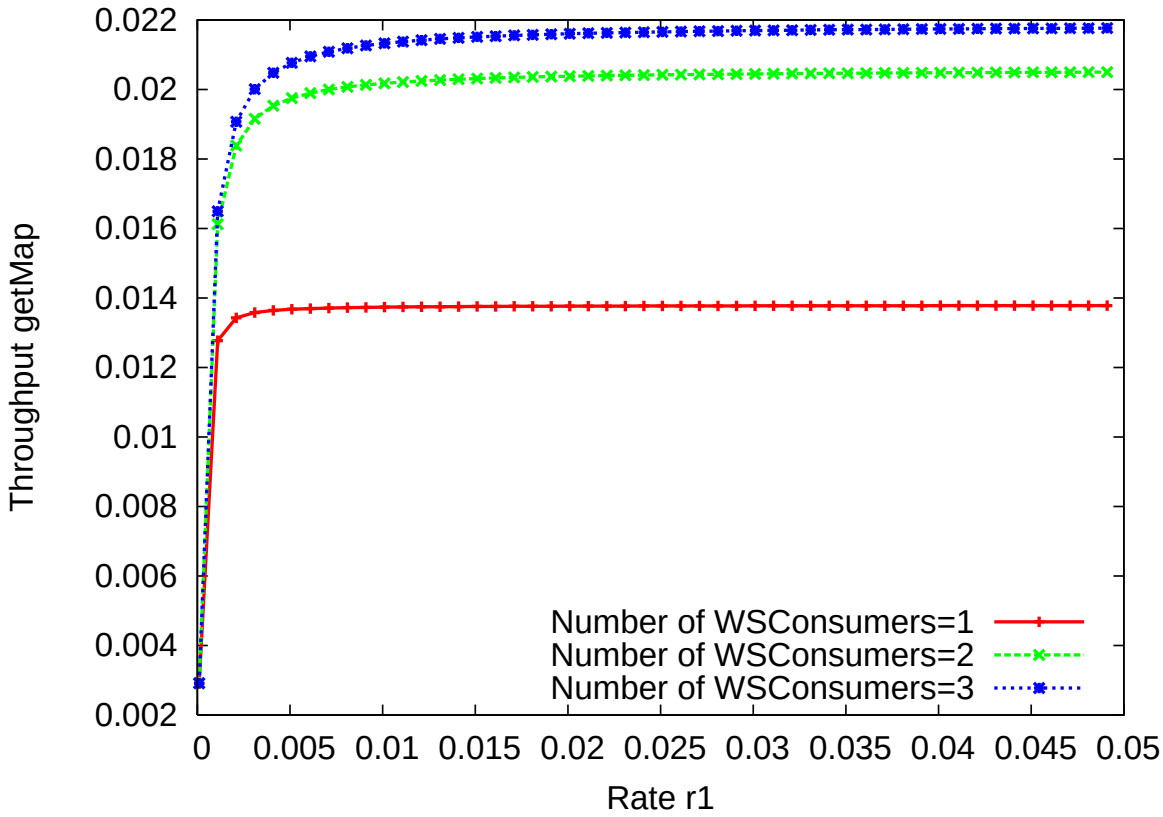


as the number of customers varies between 1 and 30 for various numbers of copies of the *WSProvider* component.

Throughput of the *getMap* action

- Under heavy load increasing the number of providers initially leads to a sharp increase in the throughput. However the gain deteriorates so that the system with four copies is just 8.7% faster than the system with three.
- In the following we settle on three copies of *WSProvider*.

Throughput of *getMap* action



as the request arrival rate (r_1) varies for differing numbers of *WSConsumer*.

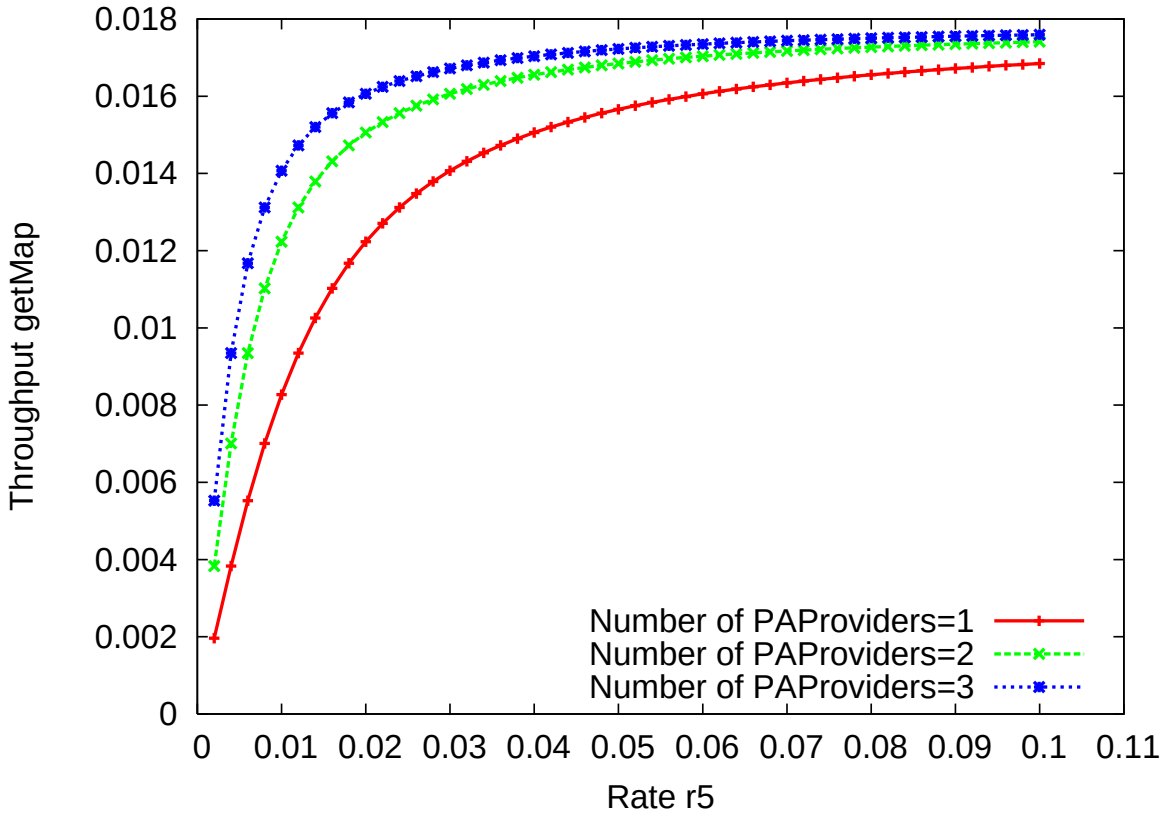
Throughput of *getMap* action

- Every line starts to plateau at approximately $r_1 = 0.010$ following an initial sharp increase. This suggests that the user is the bottle next in the system when the arrival rate is lower. Conversely, at high rates the system becomes congested.
- Whilst having two copies of *WSConsumer*, corresponding to two operating threads of control, improves performance significantly, the subsequent increase with three copies is less pronounced.
- So we set the number of copies of *WSConsumer* to 2.

Optimising the number of copies of *PAProvider*

- Here we are particularly interested in the overall impact of the rate at which the validity check is performed.
- Slower rates may mean more computationally expensive validation.
- Faster rates may involve less accuracy and lower security of the system.

Throughput of *getMap* action



as the validity check rate (r_5) varies for differing numbers of *PAProvider*.

Throughput of *getMap* action

- A sharp increase followed by a constant levelling off suggests that optimal rate values lie on the left of the plateau, as faster rates do not improve the system considerably.
- As for the optimal number of copies of *PAProvider*, deploying two copies rather than one can increase the quality of service of the overall system.
- With a similar approach as previously discussed, the modeller may want to consider the trade-off between the cost of adding a third copy and the throughput increase.

An alternative design for *PAProvider*

- The original design of *PAProvider* is stateless.
- Any of its services can be called at any point, the correctness of the system being guaranteed by implementation-specific constrainsts such as session identifiers being uniquely assigned to the clients and passed as parameters of the method calls.

An alternative design for *PAProvider*

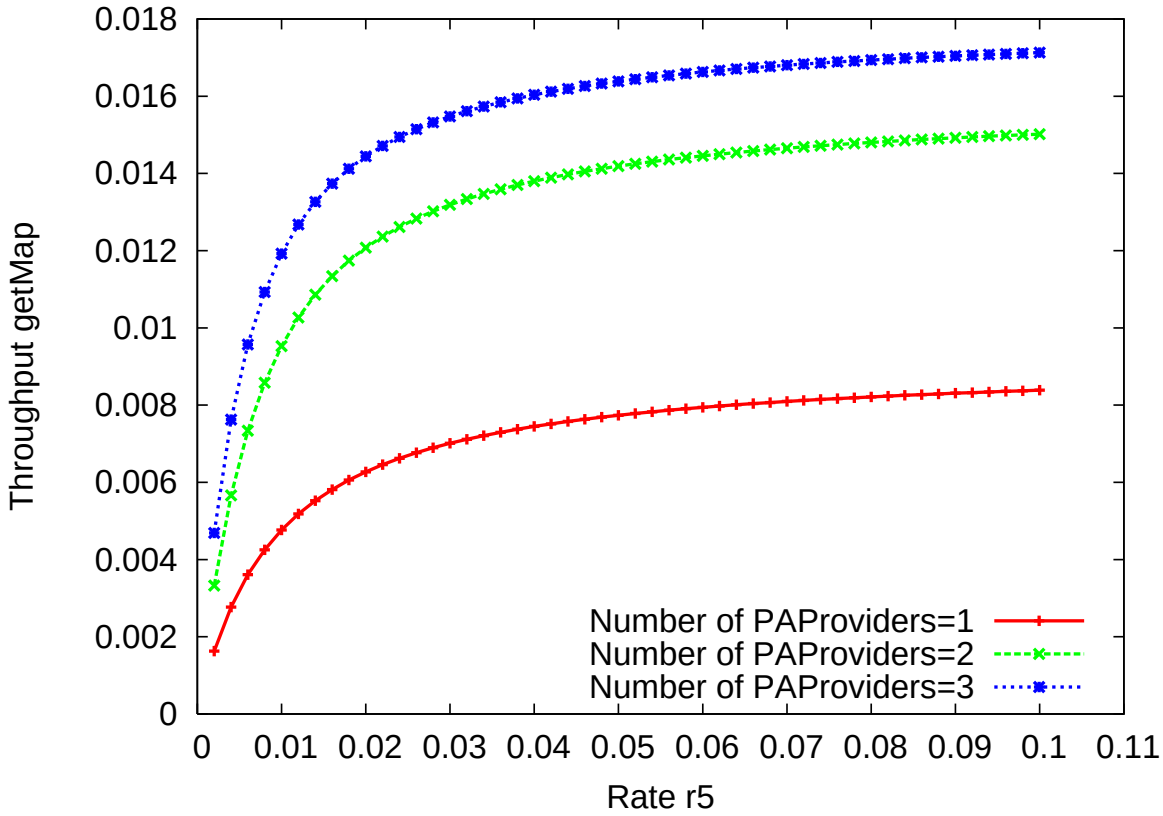
- The original design of *PAProvider* is stateless.
- Any of its services can be called at any point, the correctness of the system being guaranteed by implementation-specific constrainsts such as session identifiers being uniquely assigned to the clients and passed as parameters of the method calls.
- Alternatively we may consider a stateful implementation, modelled as a sequential component with three local states.
- This implementation has the consequence that there can never be more than N_{PAP} *WSProvider* which have started a session with a *PAProvider*

Component *PAProvider* — Stateful Version

It maintains a thread for each session and carries out the validity check on behalf of the Web Service Provider.

$$
\begin{aligned}
PAProvider &\stackrel{def}{=} (startSession, \top).PAProvider_2 \\
PAProvider_2 &\stackrel{def}{=} (checkValid, r_5).PAProvider_3 \\
PAProvider_3 &\stackrel{def}{=} (stopSession, \top).PAProvider
\end{aligned}
$$

Throughput of *getMap* action



as the validity check rate (r_5) varies for differing numbers of *PAProvider* (stateful version).

Throughput of *getMap* action

- In this case the incremental gain in adding more copies has become more marked.
- However, the modeller may want to prefer the original version, as three copies of the stateful provider deliver about as much as the throughput of only one copy of the stateless implementation.

Outline

Some concluding remarks...

The incorporation of stochastic quantitative information into process algebras has been extremely fruitful:

Some concluding remarks...

The incorporation of stochastic quantitative information into process algebras has been extremely fruitful:

- For performance modelling the rigour of the formal description technique has had benefits for both practice and theory, and lead to enhanced analysis capabilities.

Acknowledgements

The PEPA project has been funded by SERC, EPSRC and the CEC. In particular Jane Hillston and Mirco Tribastone are supported by the SENSORIA project

We would like to thank our co-authors, Allan Clark and Stephen Gilmore for help with these slides.

Thank you