Runtime Instrumentation for Reactive Components

PLAID 2024 · L. Aceto · **D. P. Attard** · A. Francalanza · A. Ingólfsdóttir

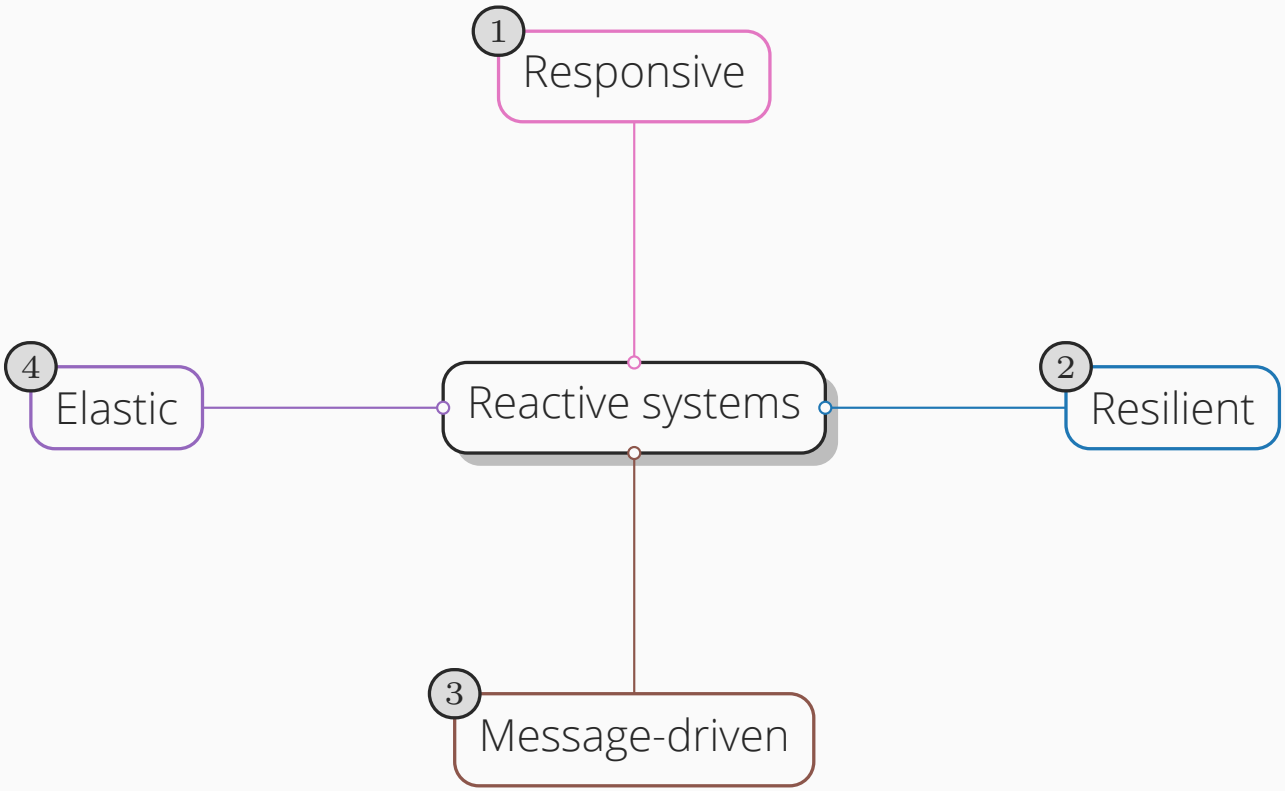
University of Glasgow

L-Università ta' Malta

Háskólinn í Reykjavík
Reykjavik University

GSSI
GRAN SASSO SCIENCE INSTITUTE

Reactive systems



Runtime monitoring... Why?

> *Understanding system behaviour requires the system to run*

profiling · resource usage analysis · security audit trails

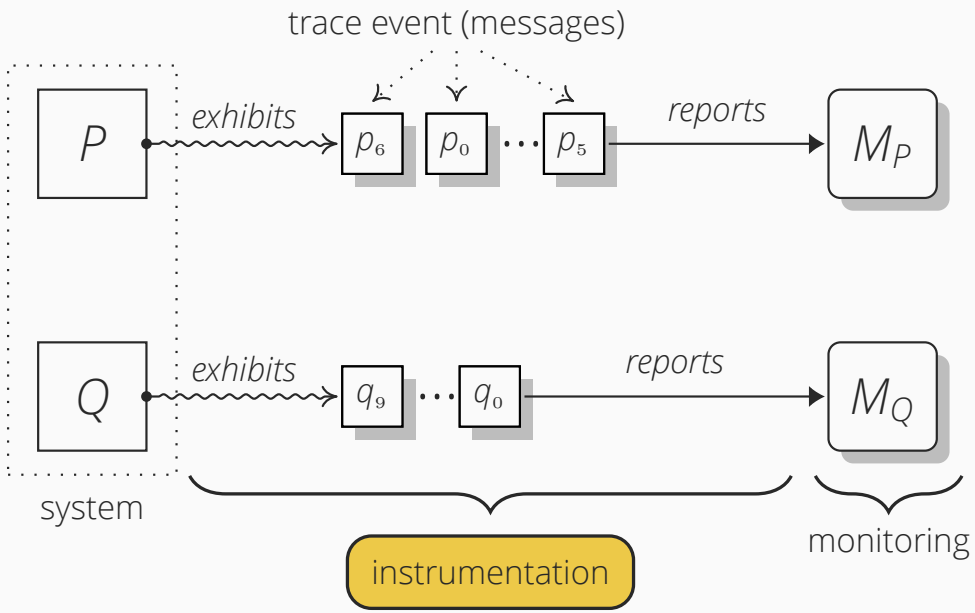
OR

> *Correctness of system is hard to analyse statically*

debugging · information flow · runtime verification

Runtime monitoring... How?

> *Extract runtime information and report it to monitors*



Roadmap

1. Runtime monitoring and verification for reactive systems
2. RIARC and how it works
3. Evaluating RIARC and results
4. Recap

Runtime monitoring requirements

> *Instrumentation **must** preserve the reactiveness of the system*

Runtime monitoring requirements

> *Instrumentation **must** preserve the reactiveness of the system*

Low overhead preserves the Responsive attribute

Independent failure preserves the Resilient attribute

Non-blocking preserves the Message-driven attribute

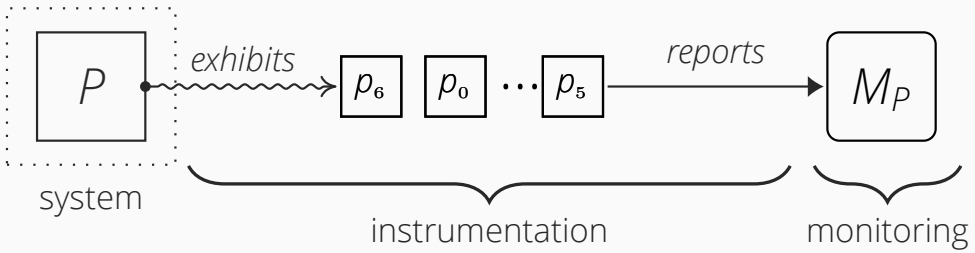
Grows and shrinks preserves the Elastic attribute

Runtime monitoring requirements

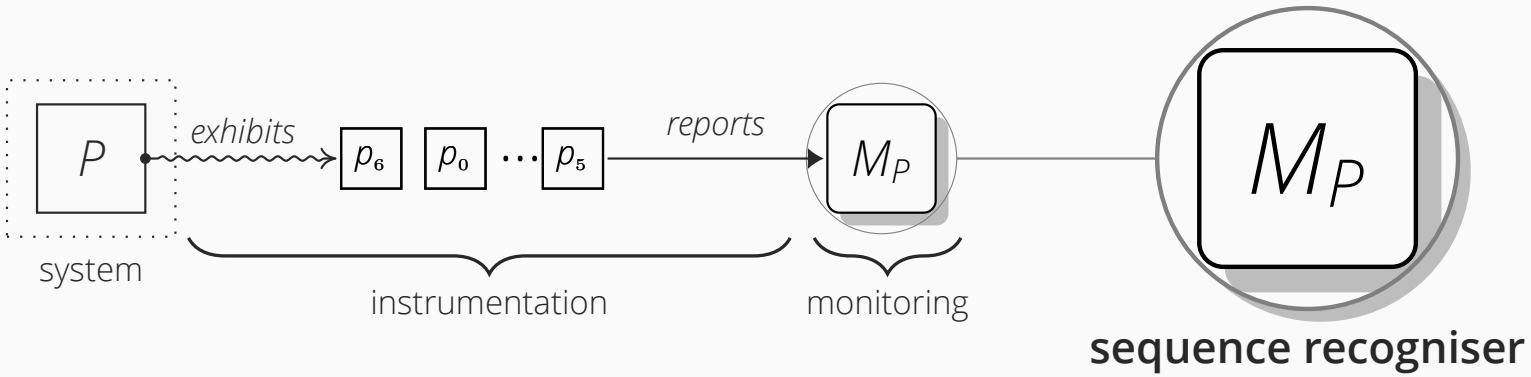
> *Instrumentation **must** preserve the reactiveness of the system*

= the instrumented system **remains** Reactive

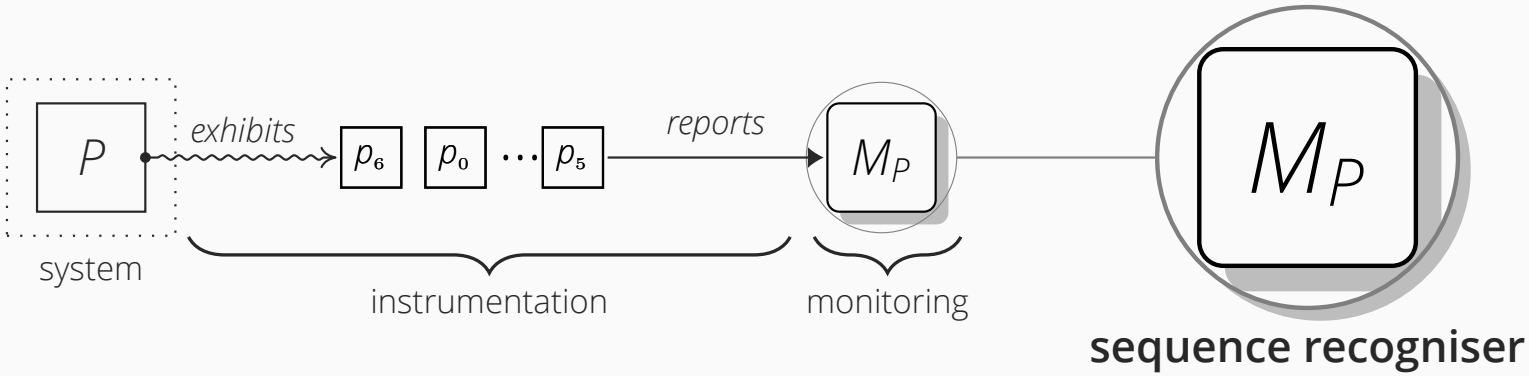
Runtime verification requirements



Runtime verification requirements



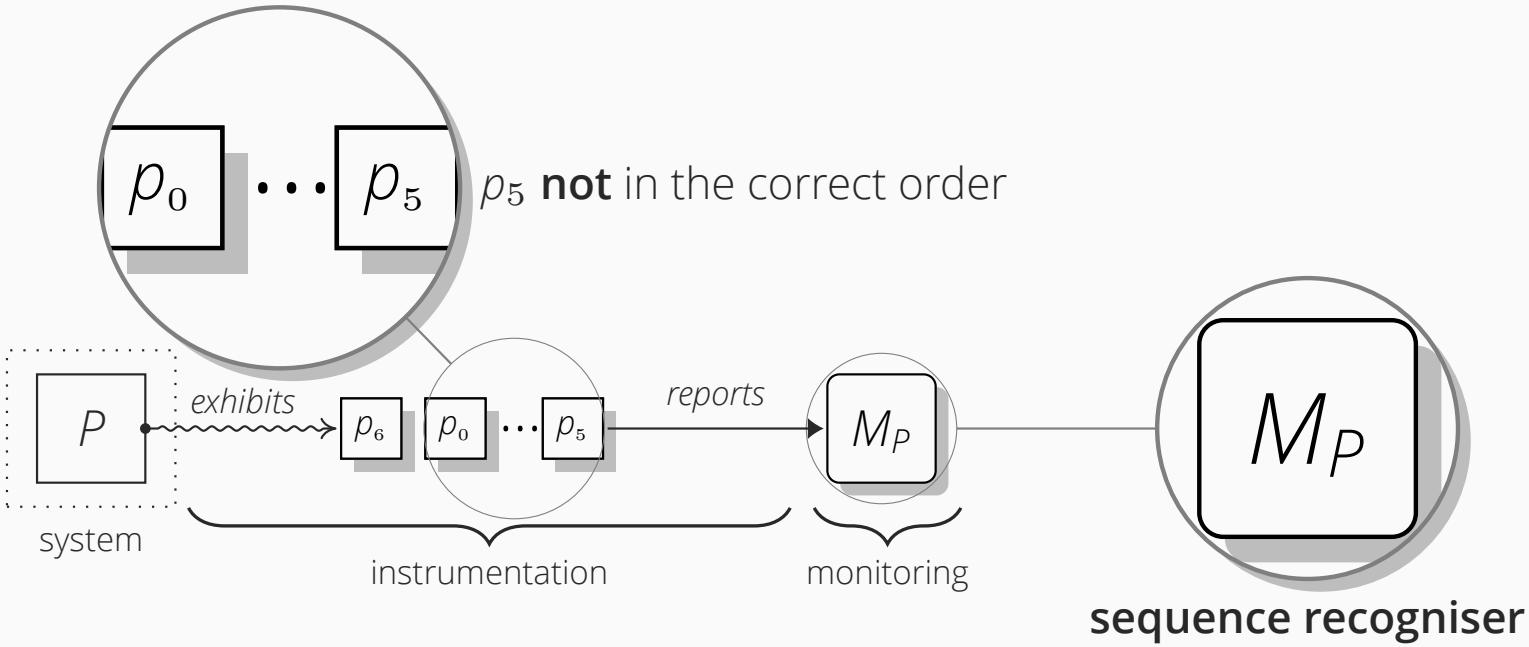
Runtime verification requirements



Trace soundness

- **Complete:** trace contains **all** the events exhibited by *P* so far
- **Consistent:** events reflect the **same order** *P* exhibits them

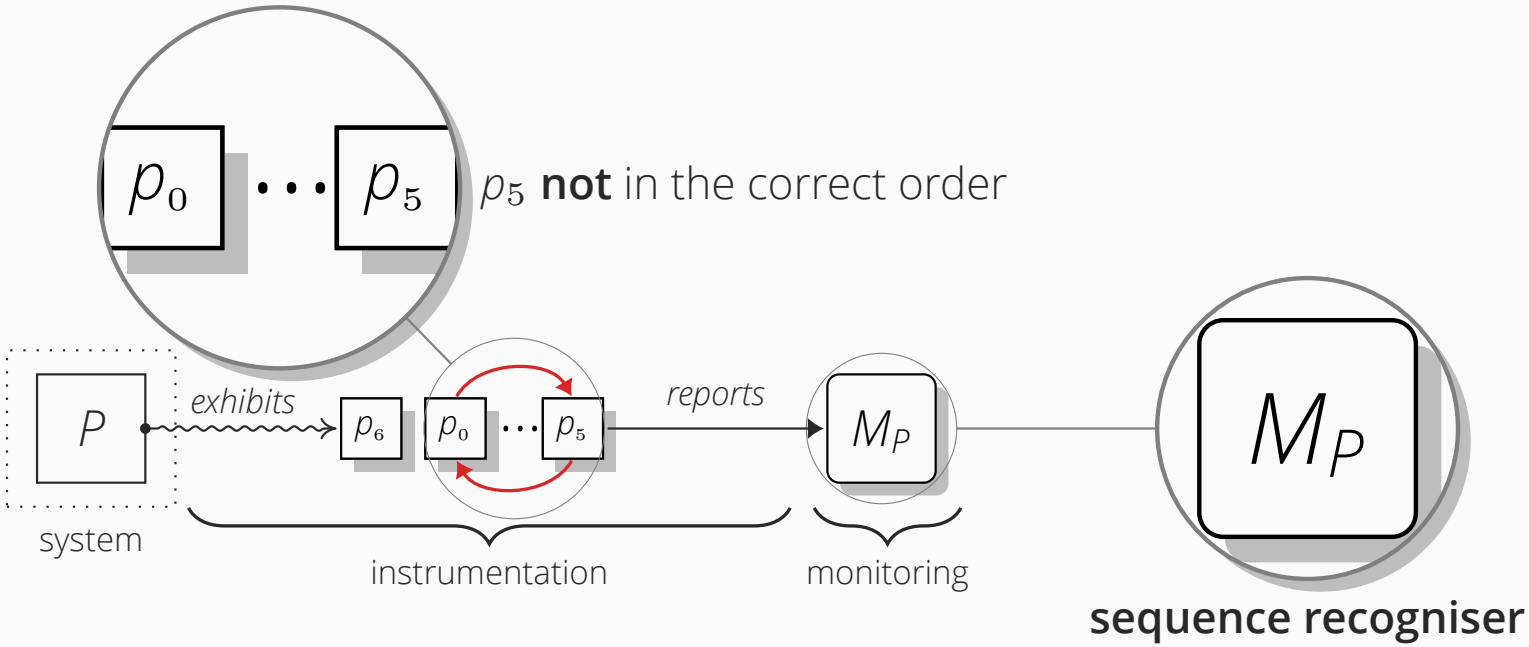
Runtime verification requirements



Trace soundness

- **Complete:** trace contains **all** the events exhibited by *P* so far
- **Consistent:** events reflect the **same order** *P* exhibits them

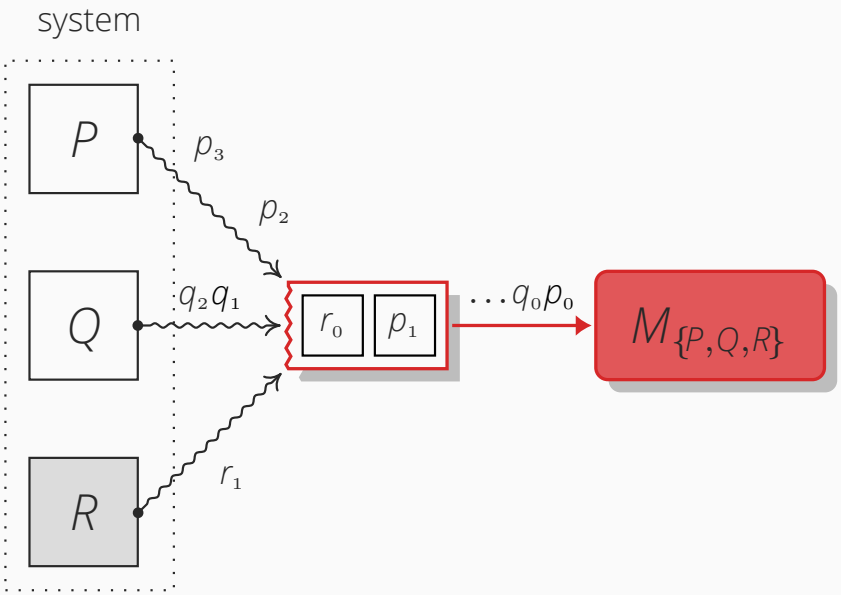
Runtime verification requirements



Trace soundness

- **Complete:** trace contains **all** the events exhibited by *P* so far
- **Consistent:** events reflect the **same order** *P* exhibits them

Centralised outline instrumentation



High overhead ✗ Responsive

Suffers from SPOF ✗ Resilient

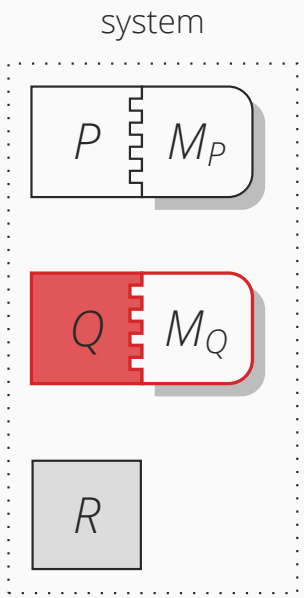
Asynchronous ✓ Message-driven

Does not shrink ✗ Elastic

Not scalable due to contention and singleton monitor

Requires **demultiplexing** for analysing events

Inline instrumentation



Typically low overhead ✓ Responsive

Embeds monitors ✗ Resilient

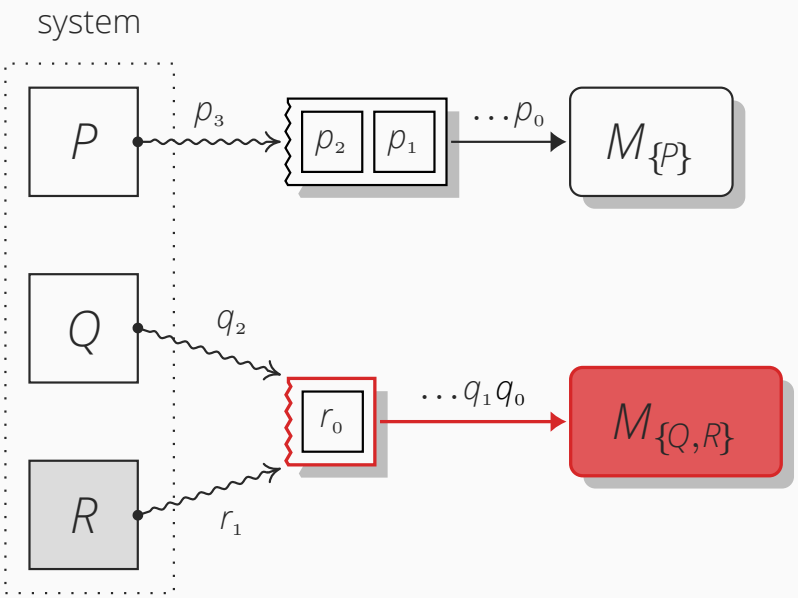
Synchronous ✗ Message-driven

Grows and shrinks ✓ Elastic

Inapplicable when code modification is not possible

Slow monitors may **impact latency**

Decentralised outline instrumentation



Feasible overhead ✓ Responsive

Isolates monitors ✓ Resilient

Asynchronous ✓ Message-driven

Grows and shrinks ✓ Elastic

Tracing uses an asynchronous **tracing infrastructure**

Dynamic outline instrumentation = **challenging** engineering

Criticism against decentralised outline monitoring

> *Decentralised outline monitoring induces **high** overhead*

Criticism against decentralised outline monitoring

> *Decentralised outline monitoring induces **high** overhead*

What do we **understand** by runtime overhead?

Most consider **execution time** as an overhead indicator

Criticism against decentralised outline monitoring

> *Decentralised outline monitoring induces **high** overhead*

What do we **understand** by runtime overhead?

~~Most consider **execution time** as an overhead indicator~~ ✗

Latency Memory consumption Scheduler usage ✓

RIARC

“ *A **reactive** decentralised outline instrumentation algorithm* ”

L. Aceto · **D. P. Attard** · A. Francalanza · A. Ingólfsdóttir · RU · GSSI · UM · U*of*G

Core idea of RIARC

> *Buffers react to **key trace events** to reorganise monitors*

Core idea of RIARC

> *Buffers react to **key trace events** to reorganise monitors*

Trace events

- spawn (⊸◇), exit (✳)
- send (!), receive (?)

Control messages

- route packet (rtd)
- detach request (dtc)

Core idea of RIARC

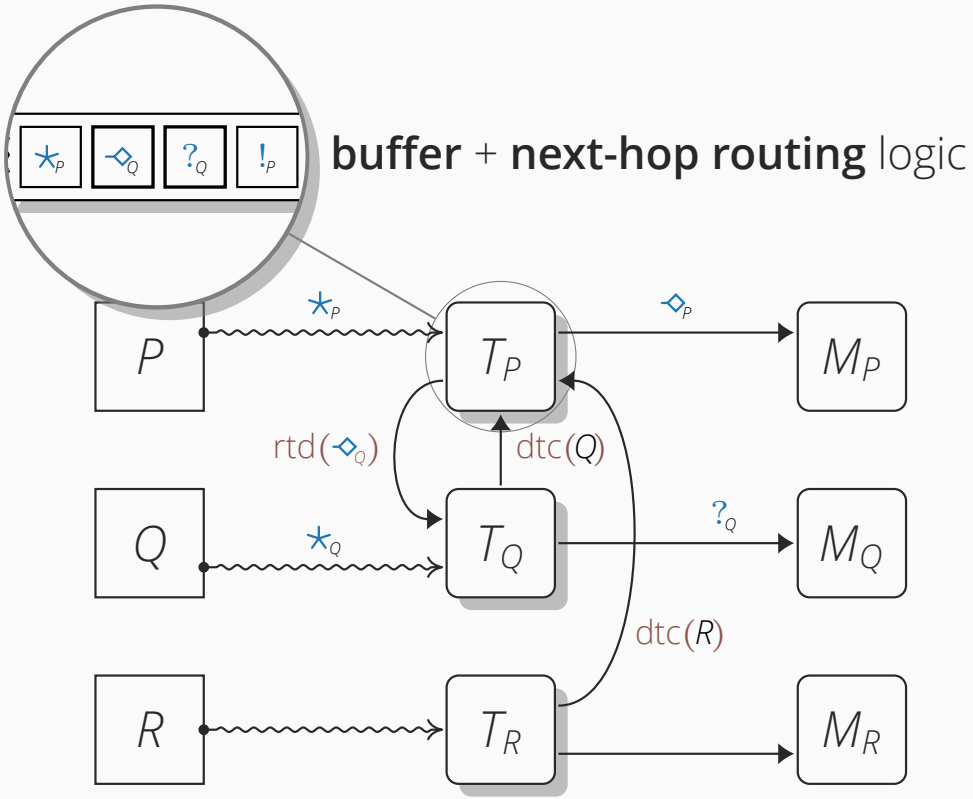
> *Buffers react to **key trace events** to reorganise monitors*

Trace events

- spawn (⊸◇), exit (✳)
- send (!), receive (?)

Control messages

- route packet (rtd)
- detach request (dtc)



Tracing assumptions RIARC makes

A1 A tracer can trace **many** processes $P_1, P_2, \ldots$

A2 A child of P **inherits** the tracer of P when spawned

A3 $\multimap$ of P gathered **before** $\multimap$, $\ast$, $!$, $?$ of a child spawned by P

A4 Any P can be traced by **at most** one tracer at a time

Tracing assumptions *RIARC* makes

A1 A tracer can trace **many** processes $P_1, P_2, \ldots$

A2 A child of P **inherits** the tracer of P when spawned

A3 $\multimap$ of P gathered **before** $\multimap$, $\ast$, $!$, $?$ of a child spawned by P

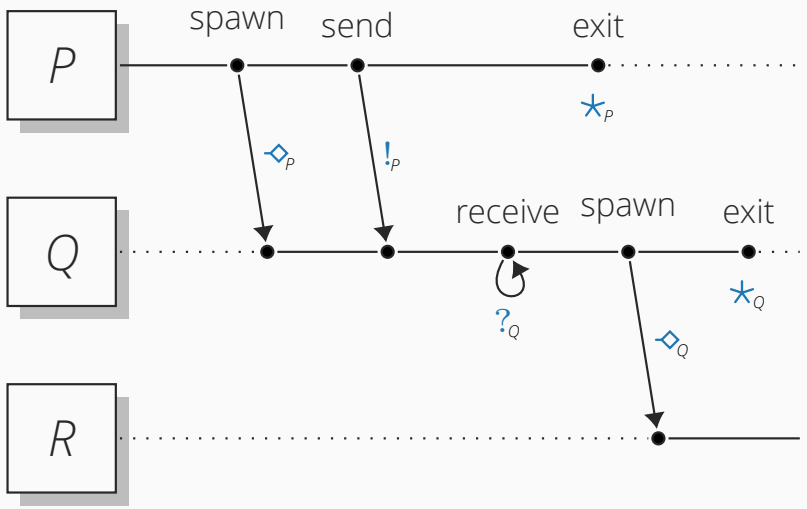
A4 Any P can be traced by **at most** one tracer at a time

Running example

A system consisting of processes P, Q, and R

P has tracer T_P, Q tracer T_Q, and R, tracer T_R

P, Q, and R are spawned in **sequence**



Growing and shrinking the set-up dynamically

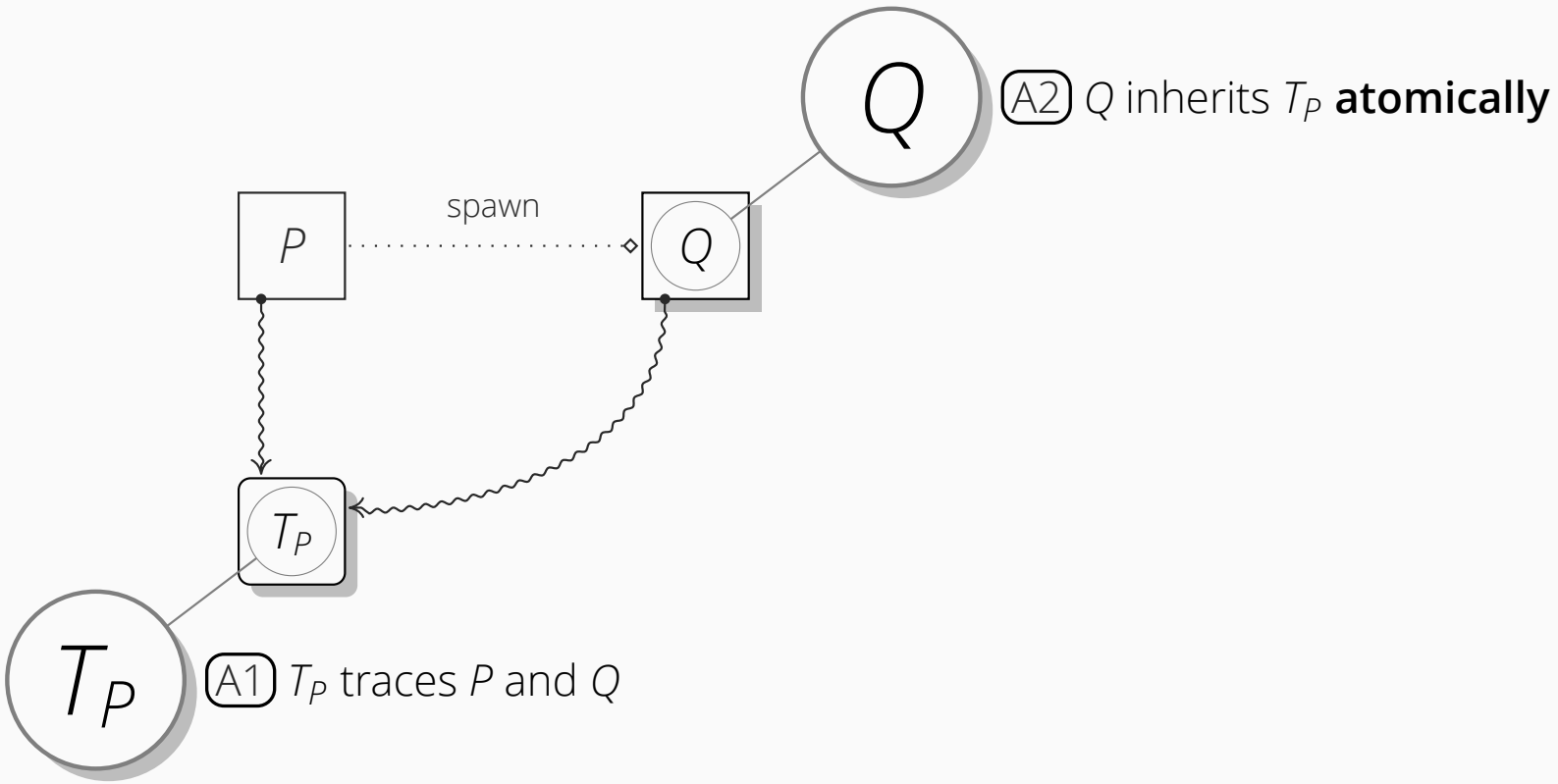
Tracers react to ⊸◇ events to **instrument** tracers

Tracers react to ✳ events **terminate** tracers not in use

Growing and shrinking the set-up dynamically

Tracers react to ⟡ events to **instrument** tracers

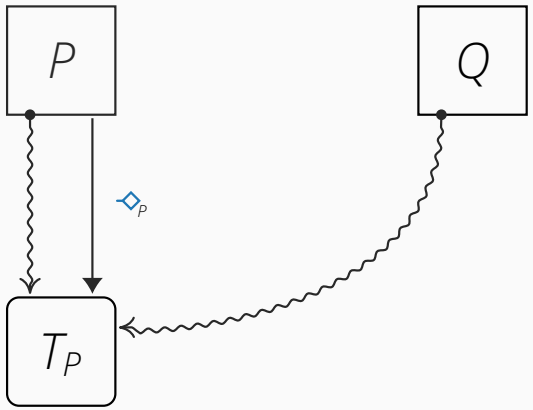
Tracers react to ✳ events **terminate** tracers not in use



Growing and shrinking the set-up dynamically

Tracers react to ⊸◇ events to **instrument** tracers

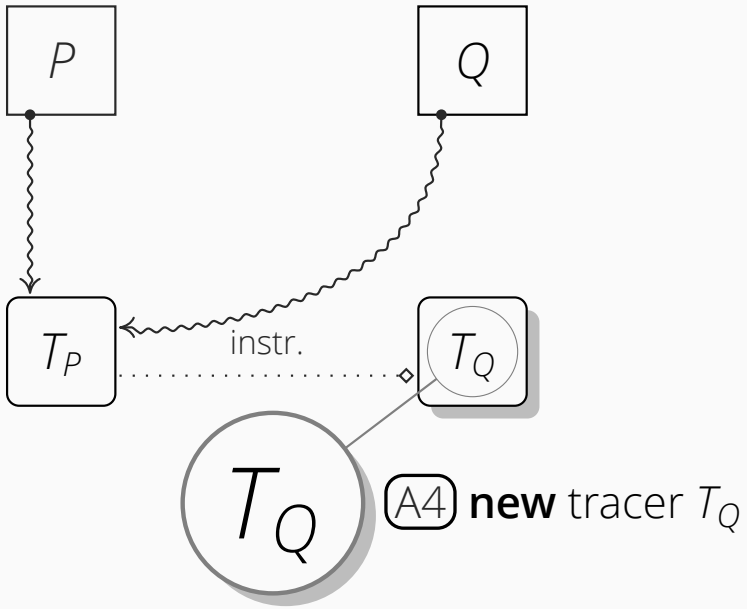
Tracers react to ✳ events **terminate** tracers not in use



Growing and shrinking the set-up dynamically

Tracers react to ⊸◇ events to **instrument** tracers

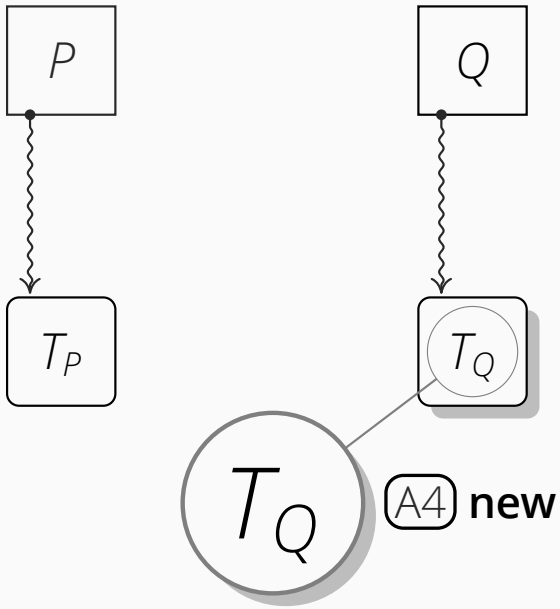
Tracers react to ✳ events **terminate** tracers not in use



Growing and shrinking the set-up dynamically

Tracers react to ⊸◇ events to **instrument** tracers

Tracers react to ✳ events **terminate** tracers not in use

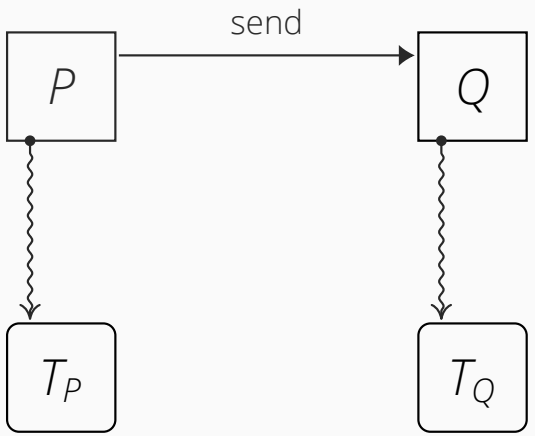


A4 **new** tracer T_Q **takes over** tracing Q from T_P

Growing and shrinking the set-up dynamically

Tracers react to ⊸◇ events to **instrument** tracers

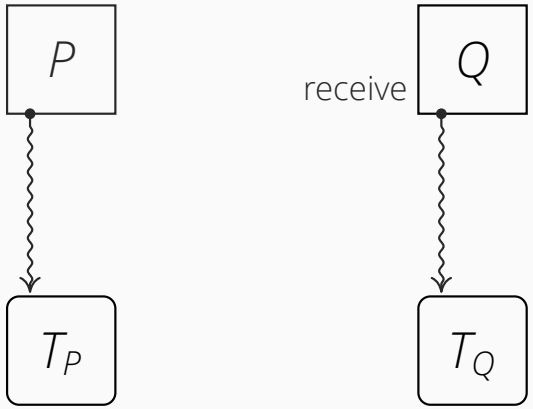
Tracers react to ✳ events **terminate** tracers not in use



Growing and shrinking the set-up dynamically

Tracers react to ⊸◇ events to **instrument** tracers

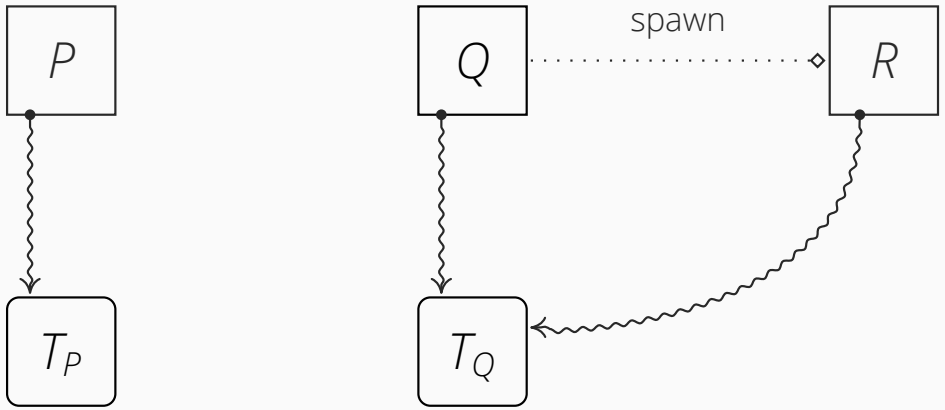
Tracers react to ✳ events **terminate** tracers not in use



Growing and shrinking the set-up dynamically

Tracers react to ⊸◇ events to **instrument** tracers

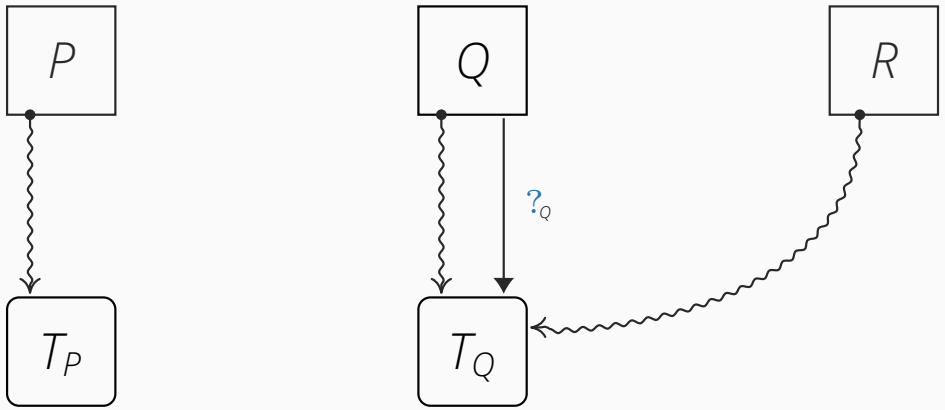
Tracers react to ✳ events **terminate** tracers not in use



Growing and shrinking the set-up dynamically

Tracers react to ⊸◇ events to **instrument** tracers

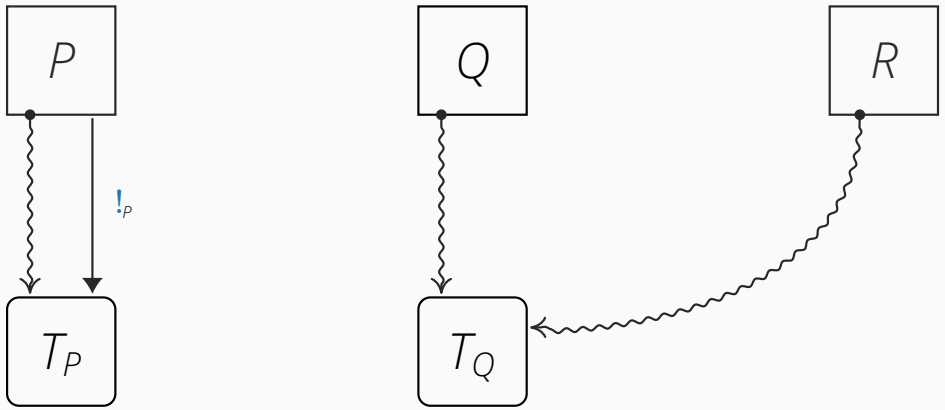
Tracers react to ✳ events **terminate** tracers not in use



Growing and shrinking the set-up dynamically

Tracers react to ⊸◇ events to **instrument** tracers

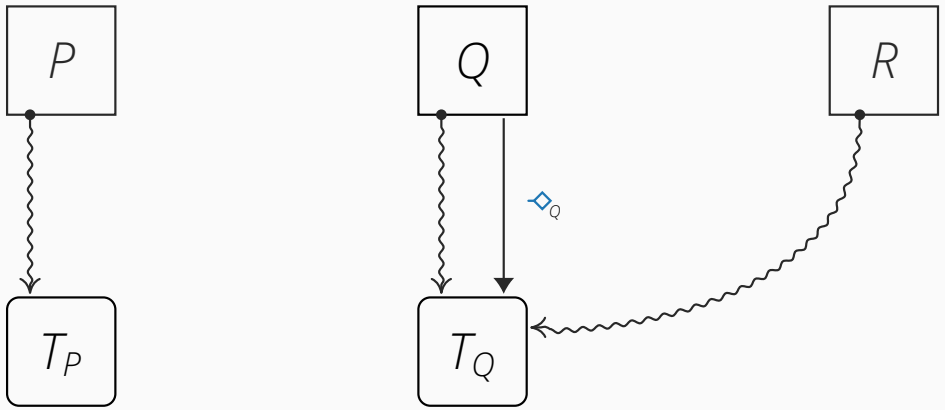
Tracers react to ✳ events **terminate** tracers not in use



Growing and shrinking the set-up dynamically

Tracers react to ⊸◇ events to **instrument** tracers

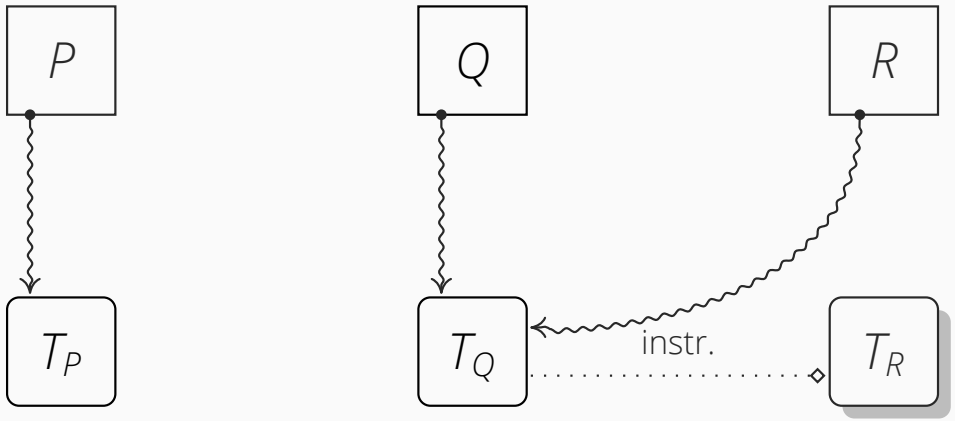
Tracers react to ✳ events **terminate** tracers not in use



Growing and shrinking the set-up dynamically

Tracers react to ⊸◇ events to **instrument** tracers

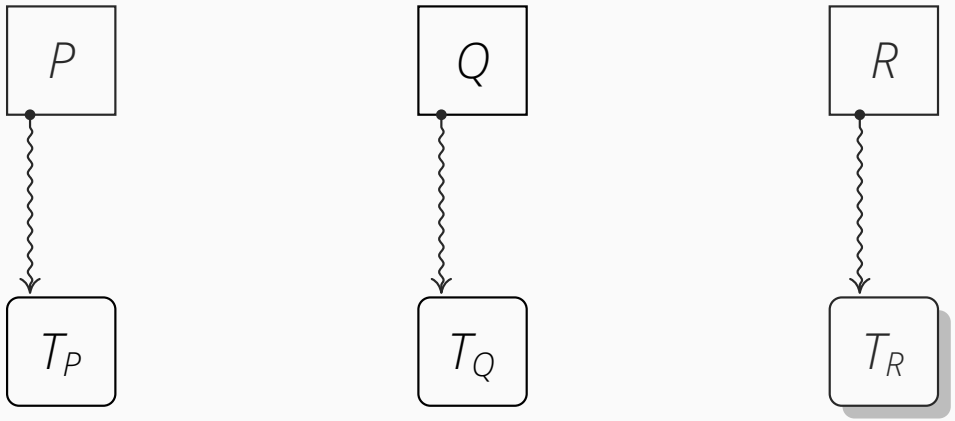
Tracers react to ✳ events **terminate** tracers not in use



Growing and shrinking the set-up dynamically

Tracers react to ⊸◇ events to **instrument** tracers

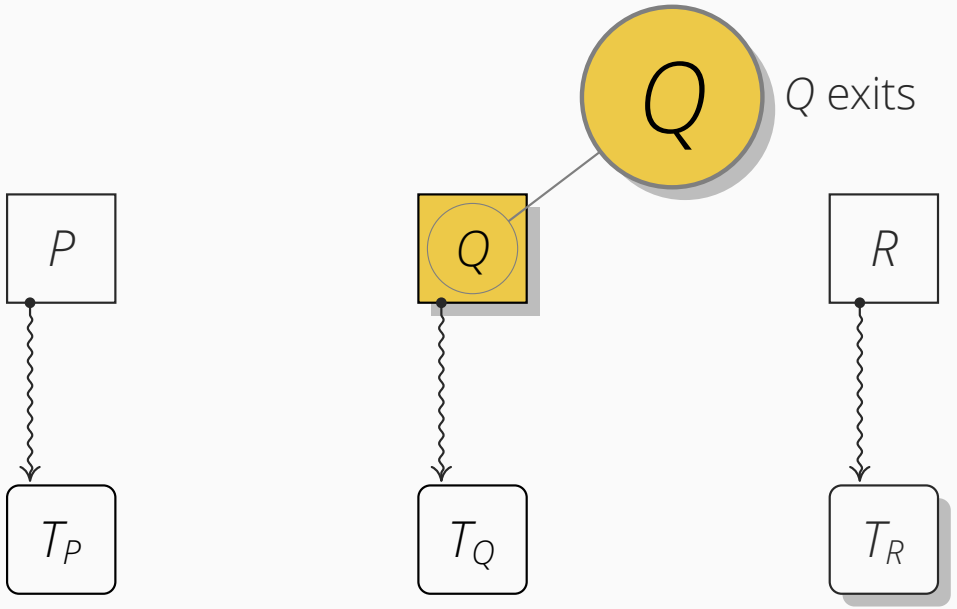
Tracers react to ✳ events **terminate** tracers not in use



Growing and shrinking the set-up dynamically

Tracers react to ⊸◇ events to **instrument** tracers

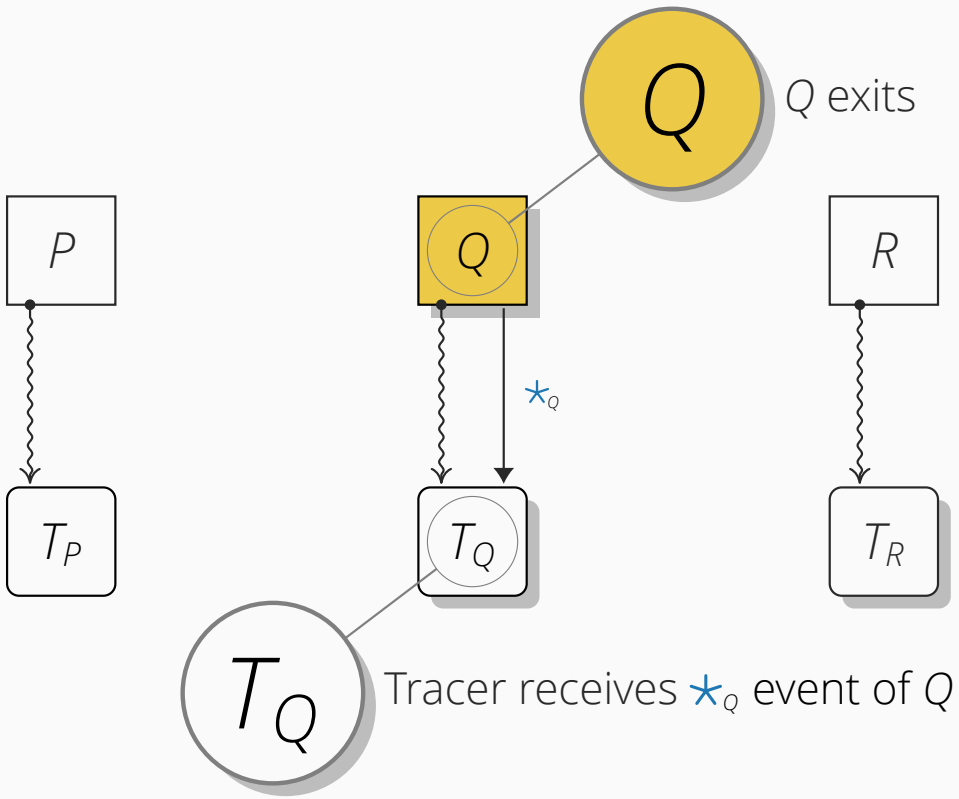
Tracers react to ✳ events **terminate** tracers not in use



Growing and shrinking the set-up dynamically

Tracers react to $\multimap\!\diamond$ events to **instrument** tracers

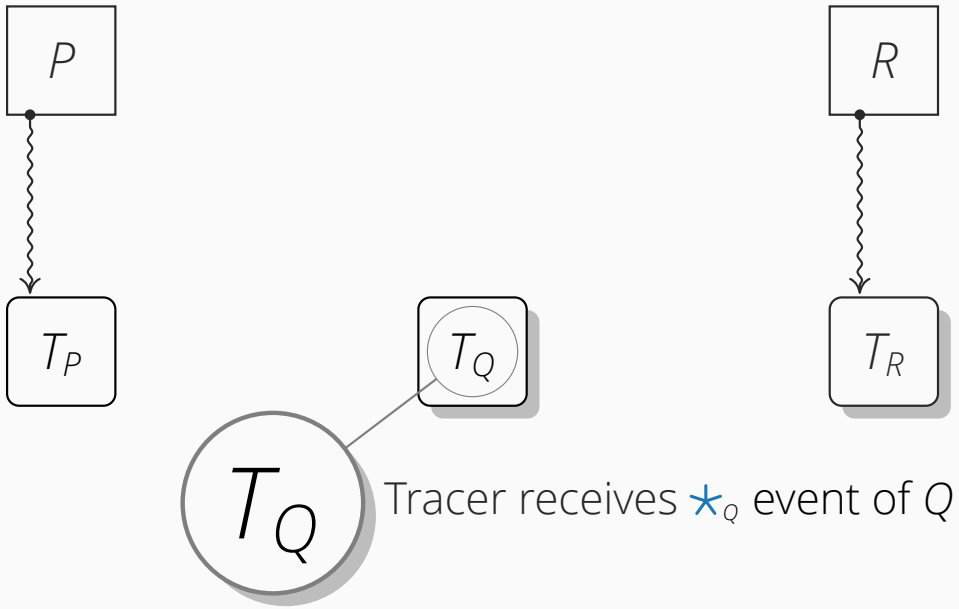
Tracers react to $\ast$ events **terminate** tracers not in use



Growing and shrinking the set-up dynamically

Tracers react to $\multimap\!\diamond$ events to **instrument** tracers

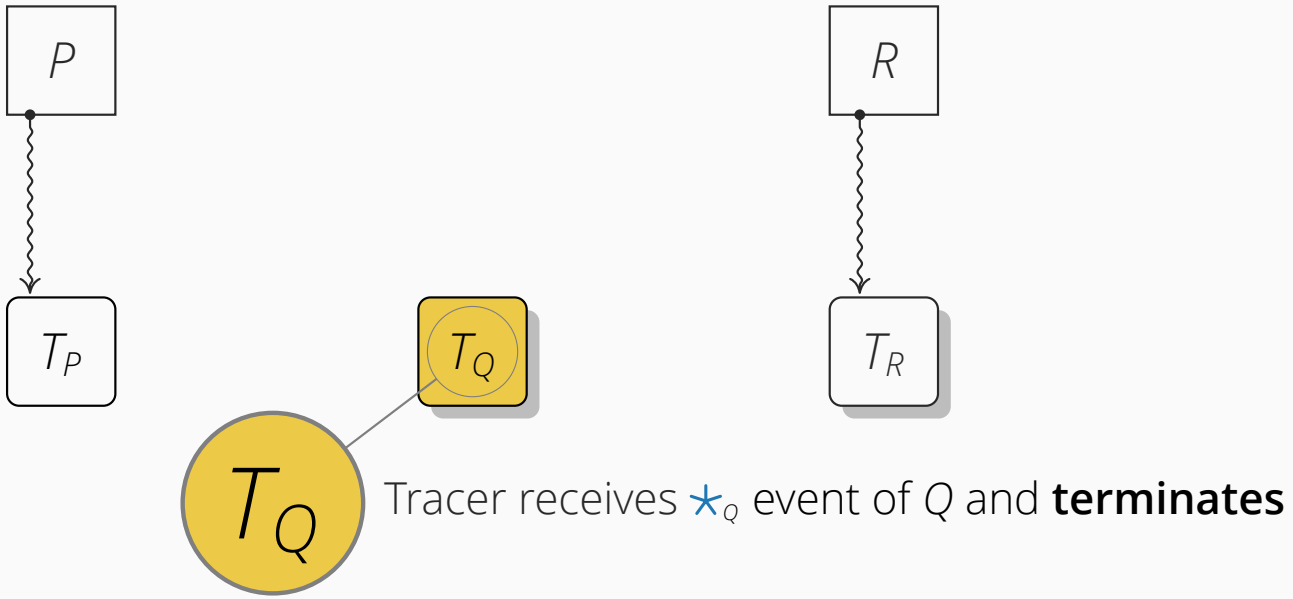
Tracers react to $\ast$ events **terminate** tracers not in use



Growing and shrinking the set-up dynamically

Tracers react to $\multimap\!\diamond$ events to **instrument** tracers

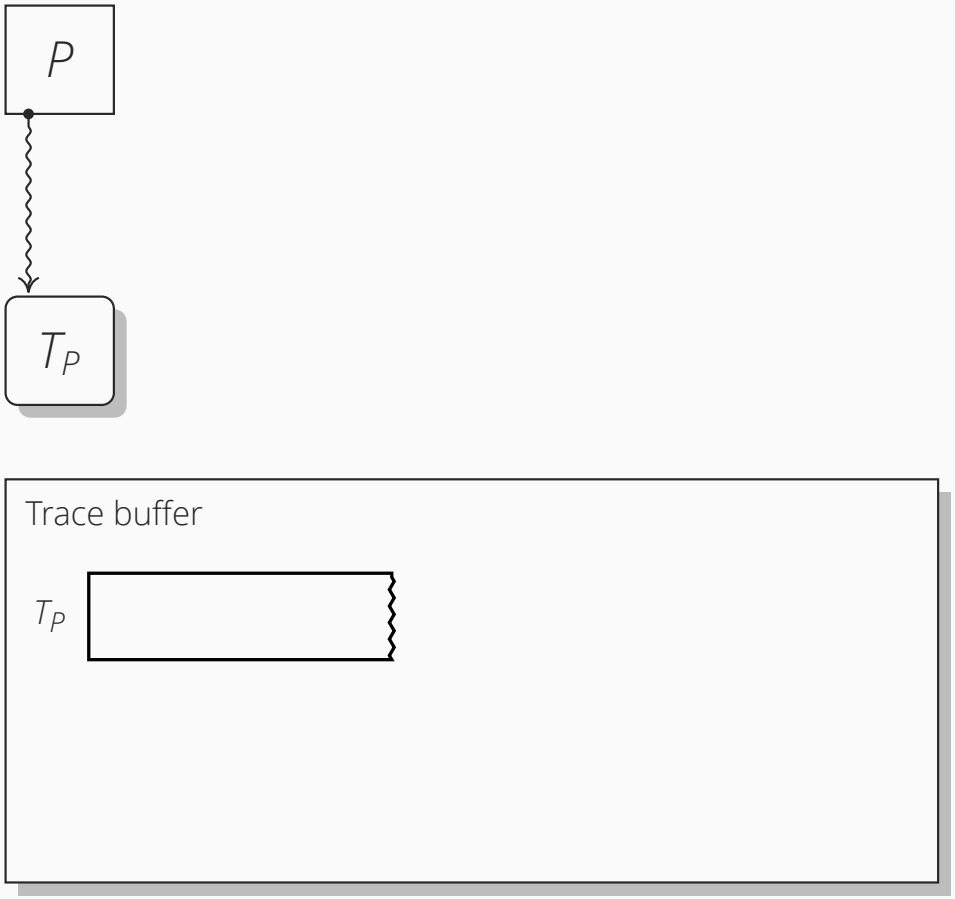
Tracers react to $\ast$ events **terminate** tracers not in use



The effects of asynchronous tracing

Interleavings between system and tracers

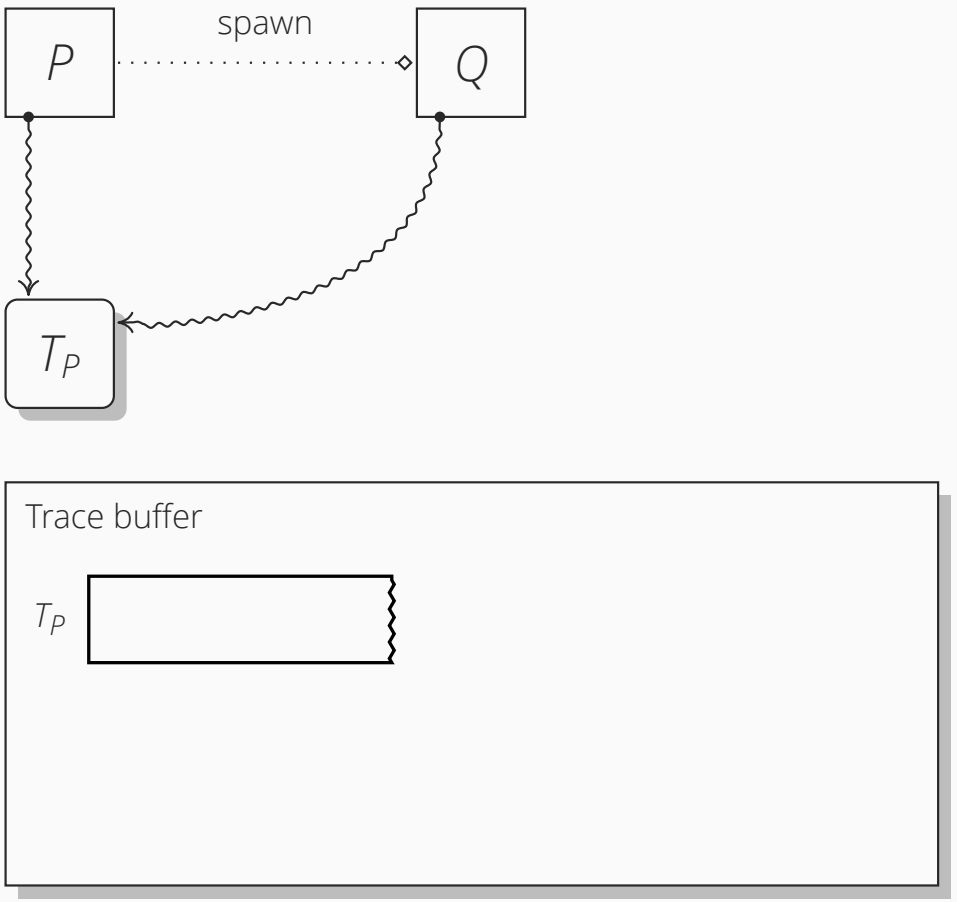
Incomplete traces



The effects of asynchronous tracing

Interleavings between system and tracers

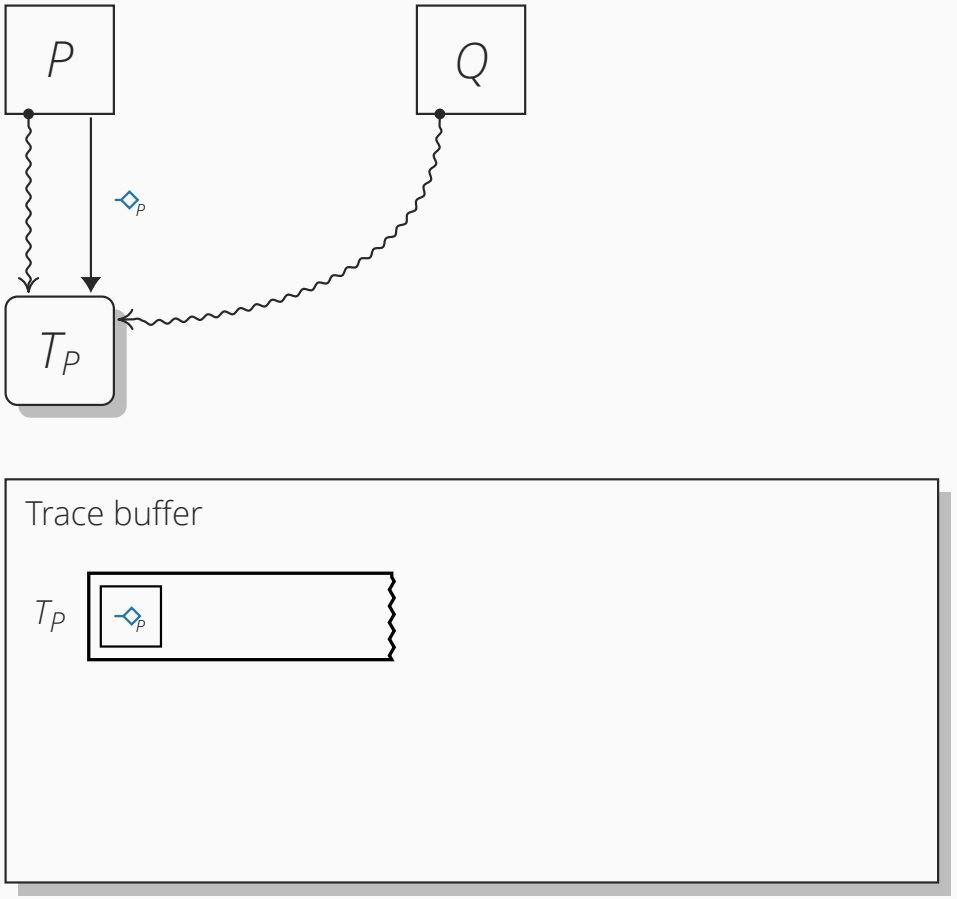
Incomplete traces



The effects of asynchronous tracing

Interleavings between system and tracers

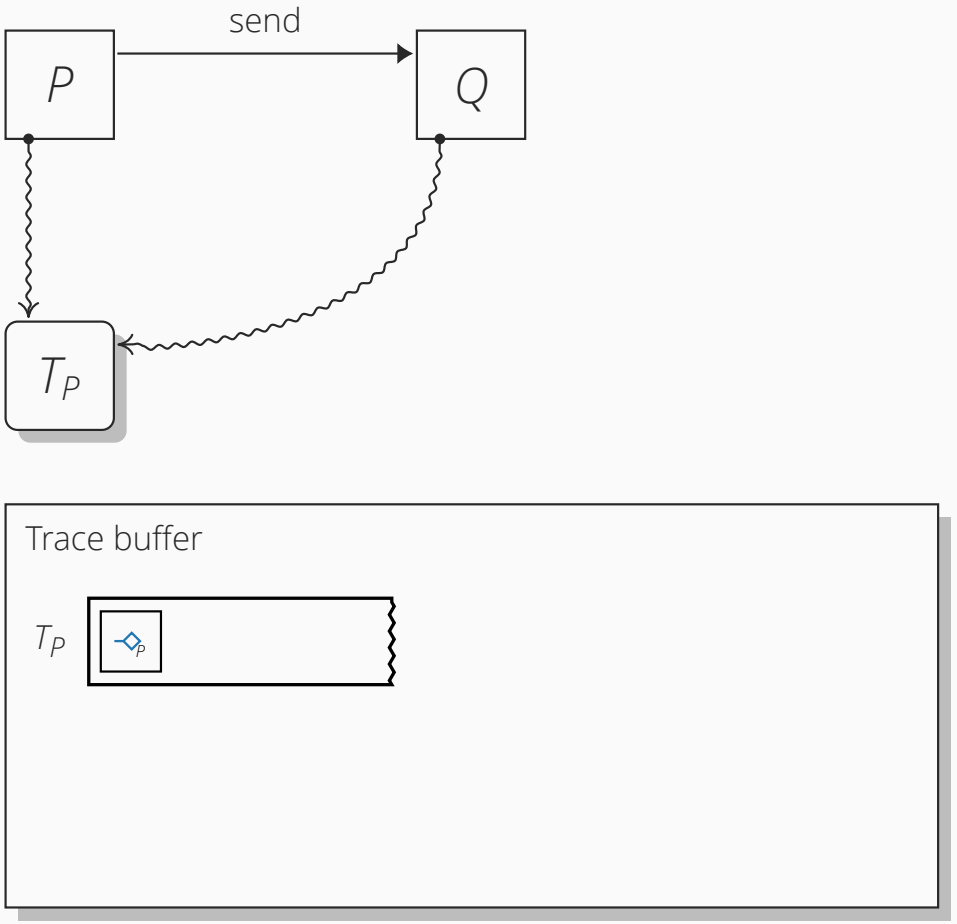
Incomplete traces



The effects of asynchronous tracing

Interleavings between system and tracers

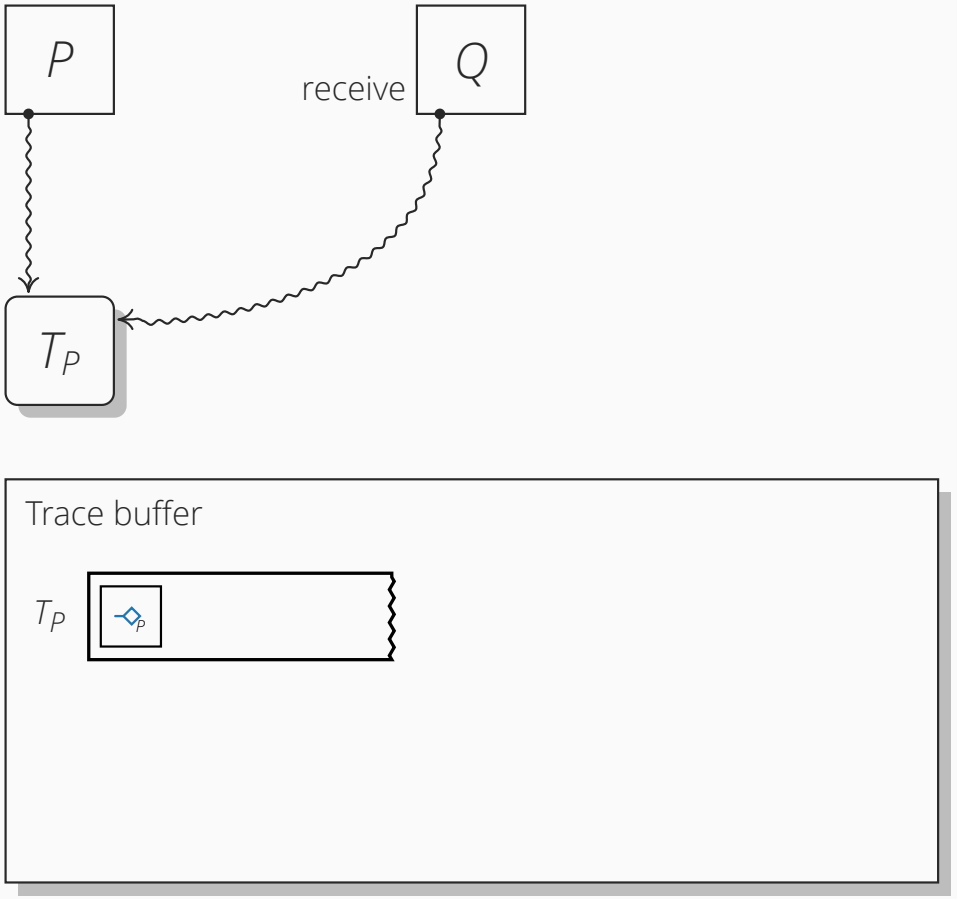
Incomplete traces



The effects of asynchronous tracing

Interleavings between system and tracers

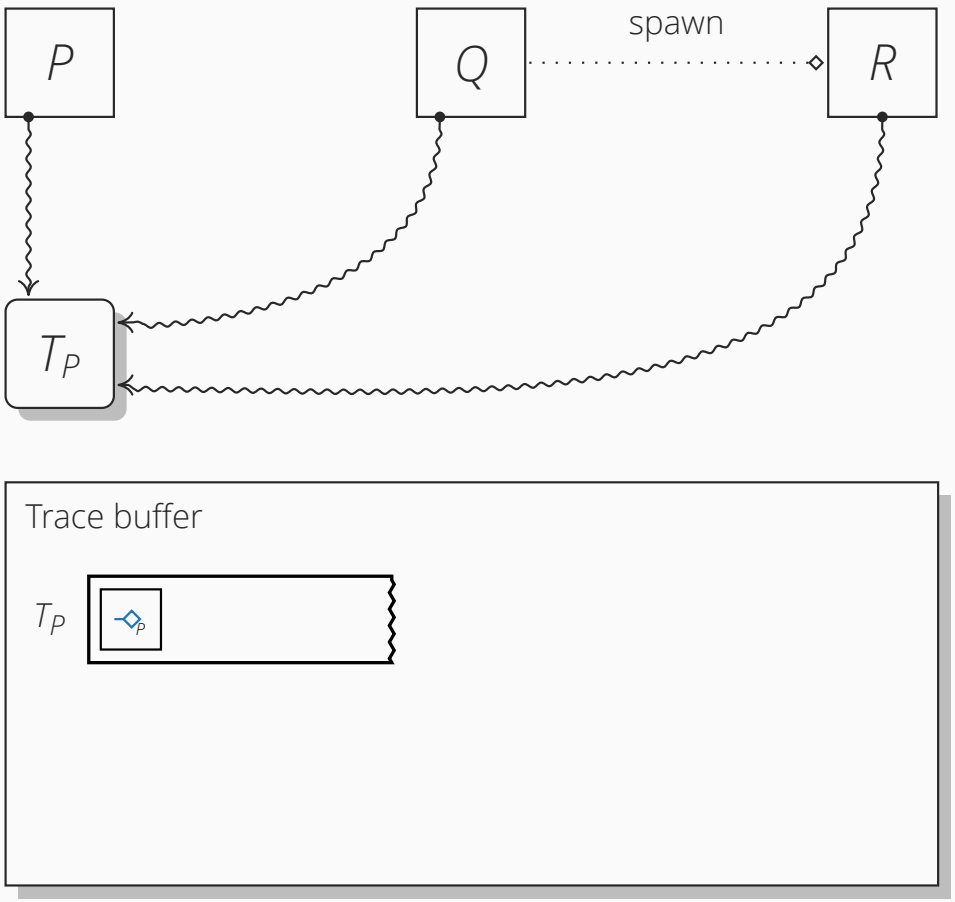
Incomplete traces



The effects of asynchronous tracing

Interleavings between system and tracers

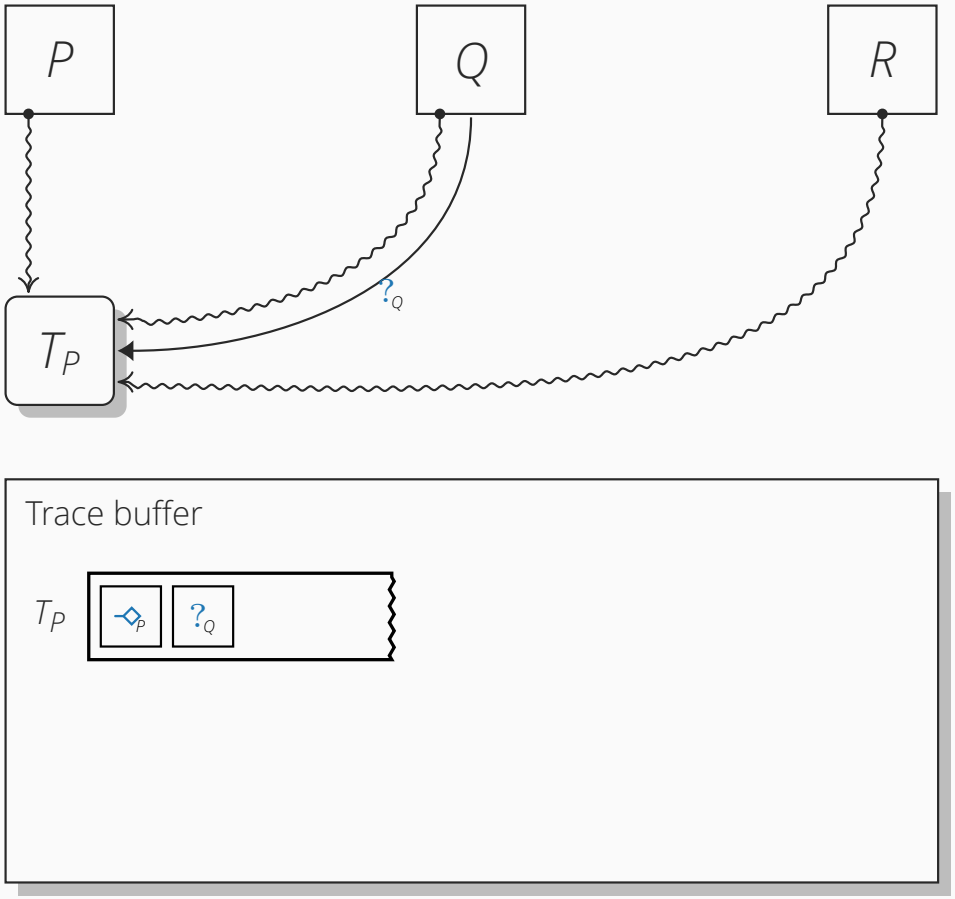
Incomplete traces



The effects of asynchronous tracing

Interleavings between system and tracers

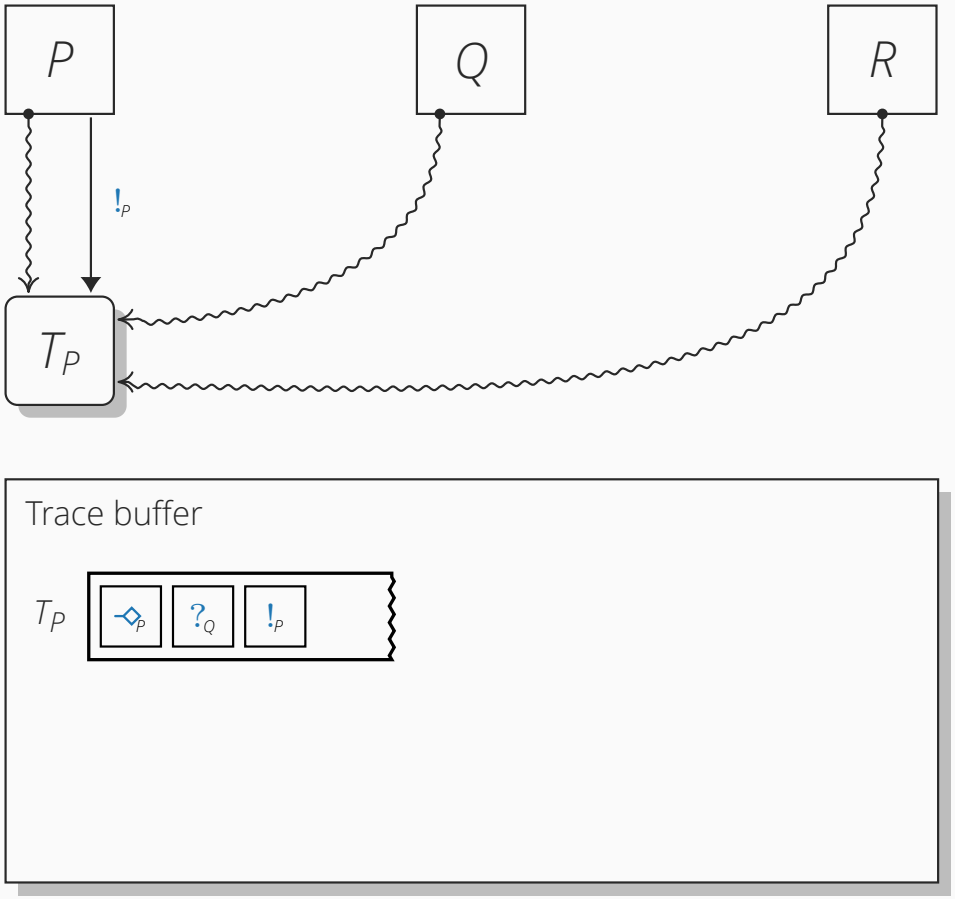
Incomplete traces



The effects of asynchronous tracing

Interleavings between system and tracers

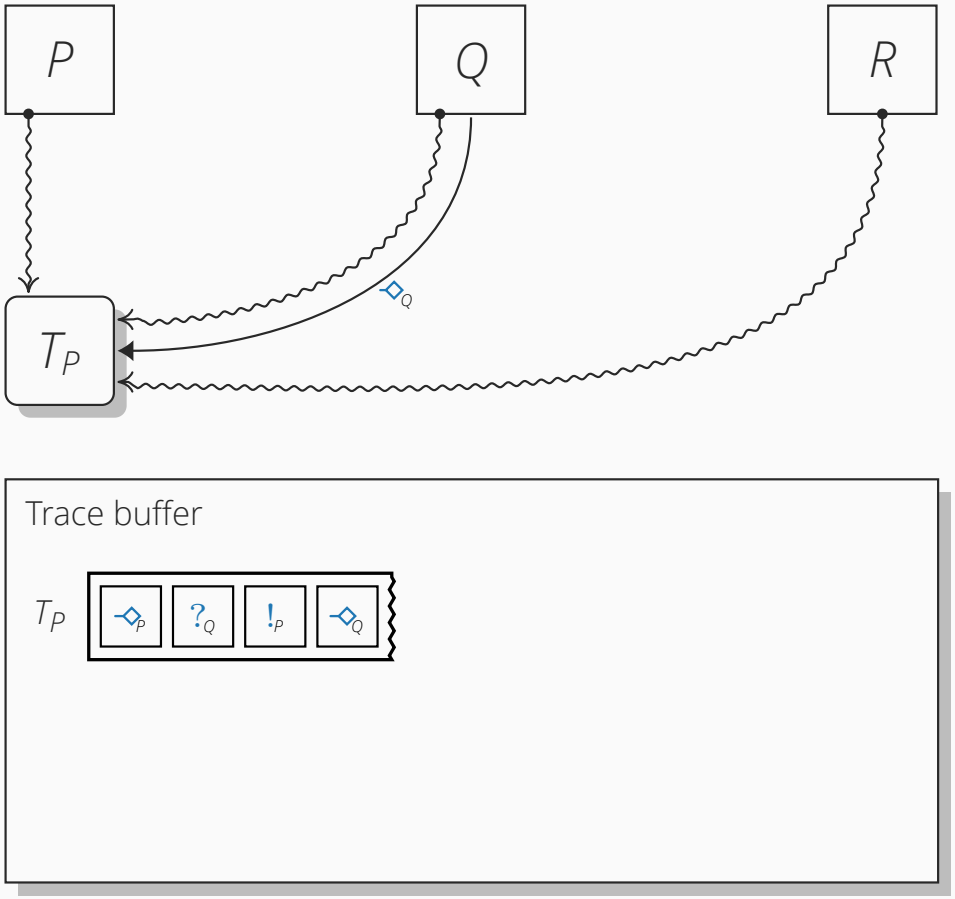
Incomplete traces



The effects of asynchronous tracing

Interleavings between system and tracers

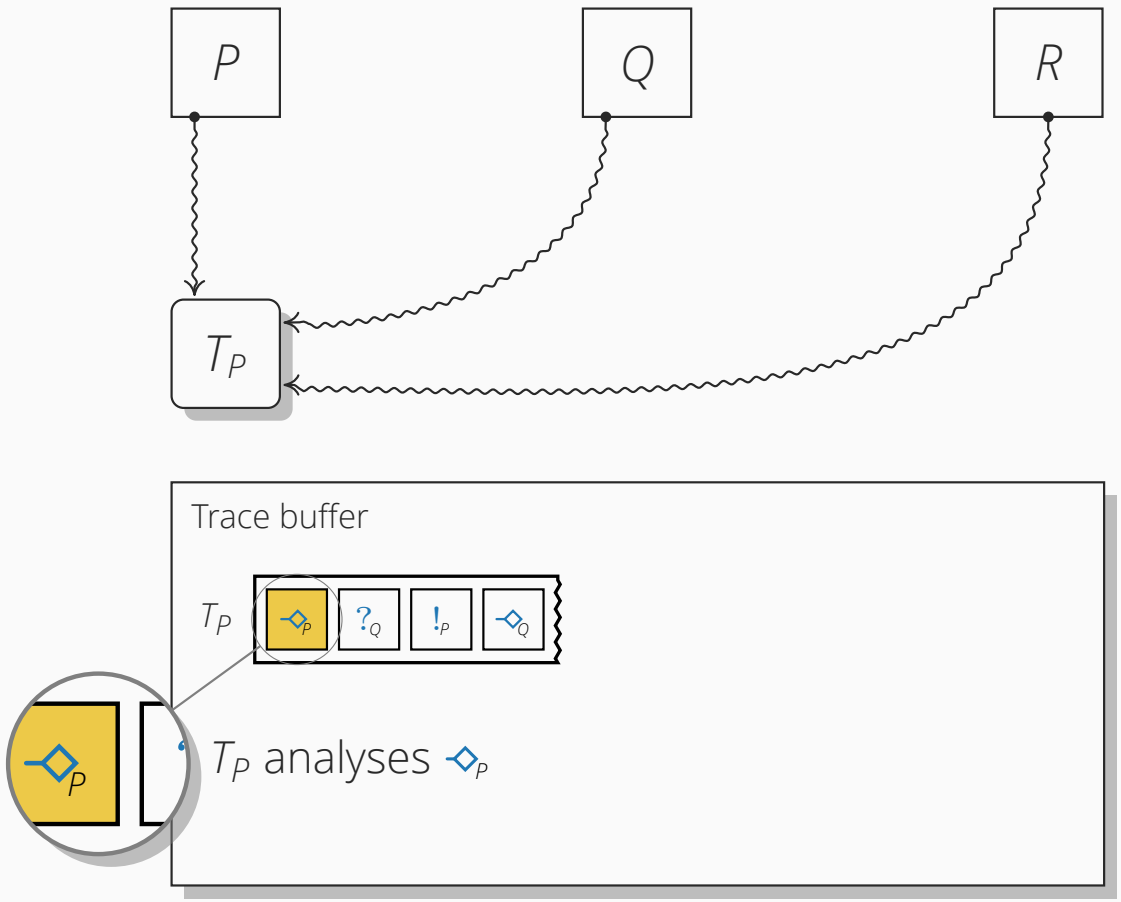
Incomplete traces



The effects of asynchronous tracing

Interleavings between system and tracers

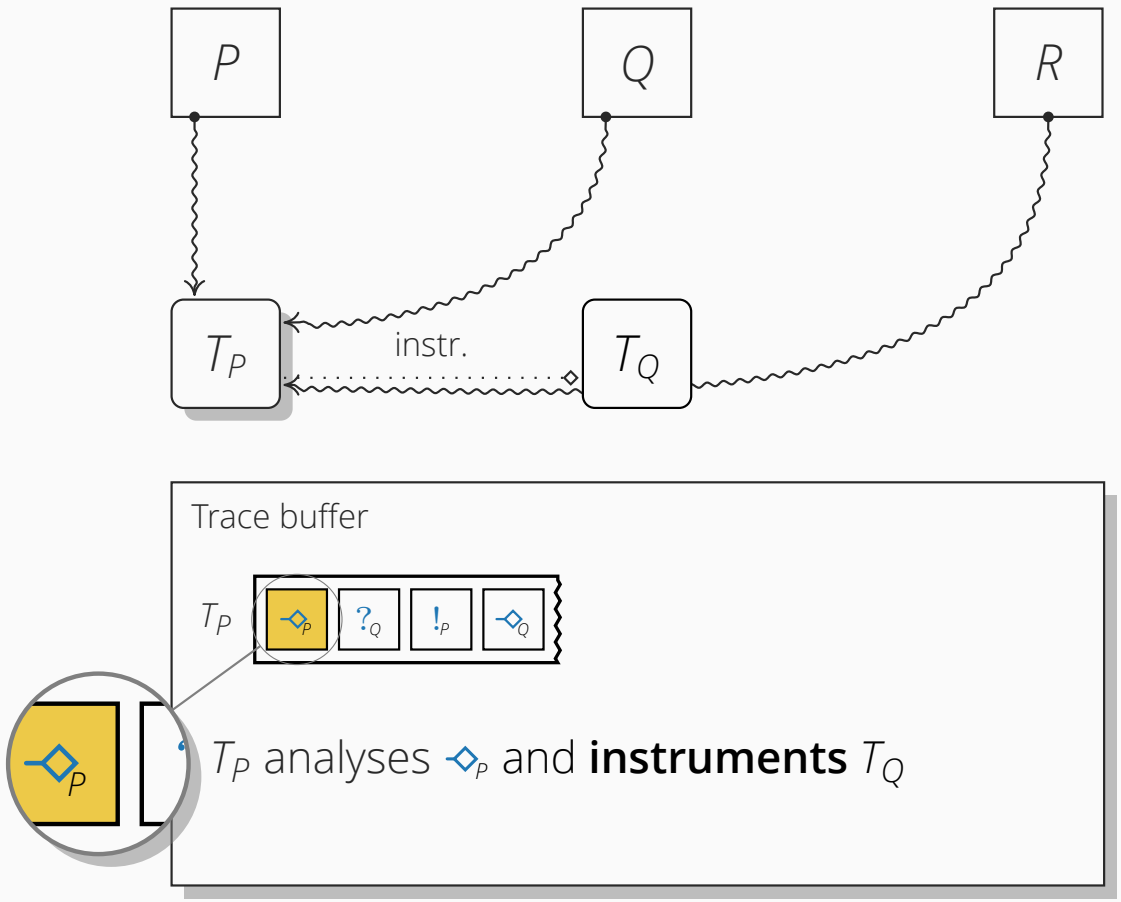
Incomplete traces



The effects of asynchronous tracing

Interleavings between system and tracers

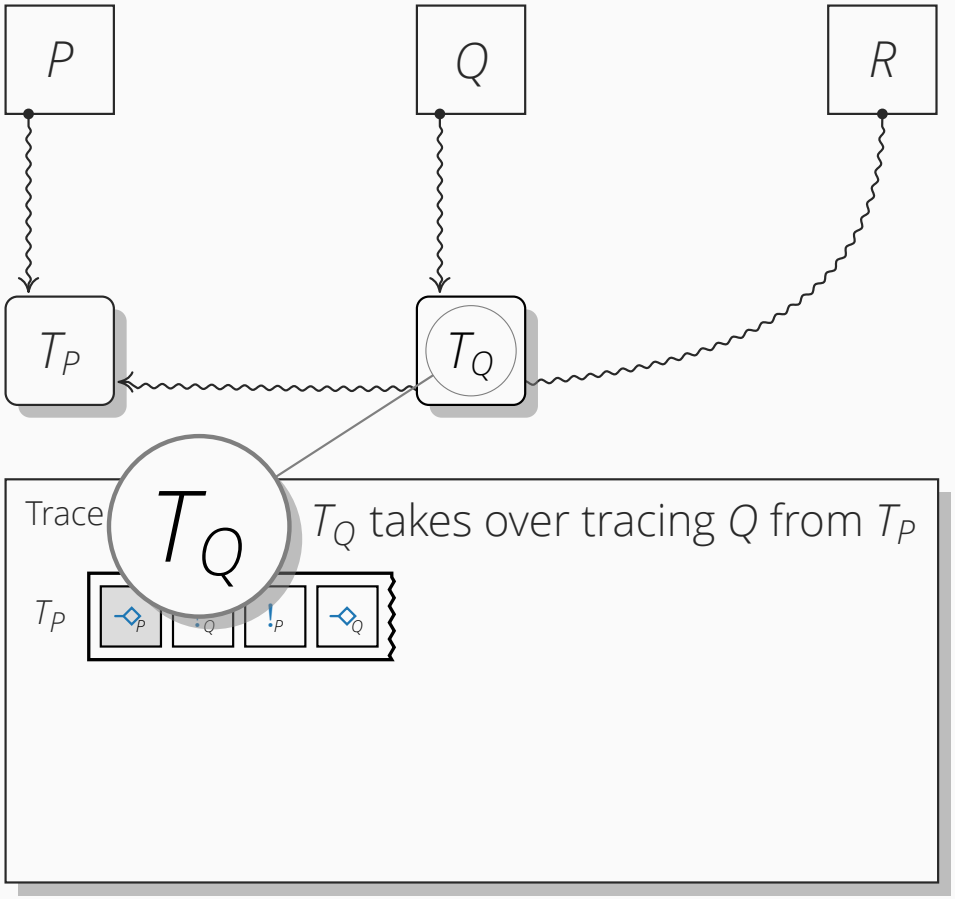
Incomplete traces



The effects of asynchronous tracing

Interleavings between system and tracers

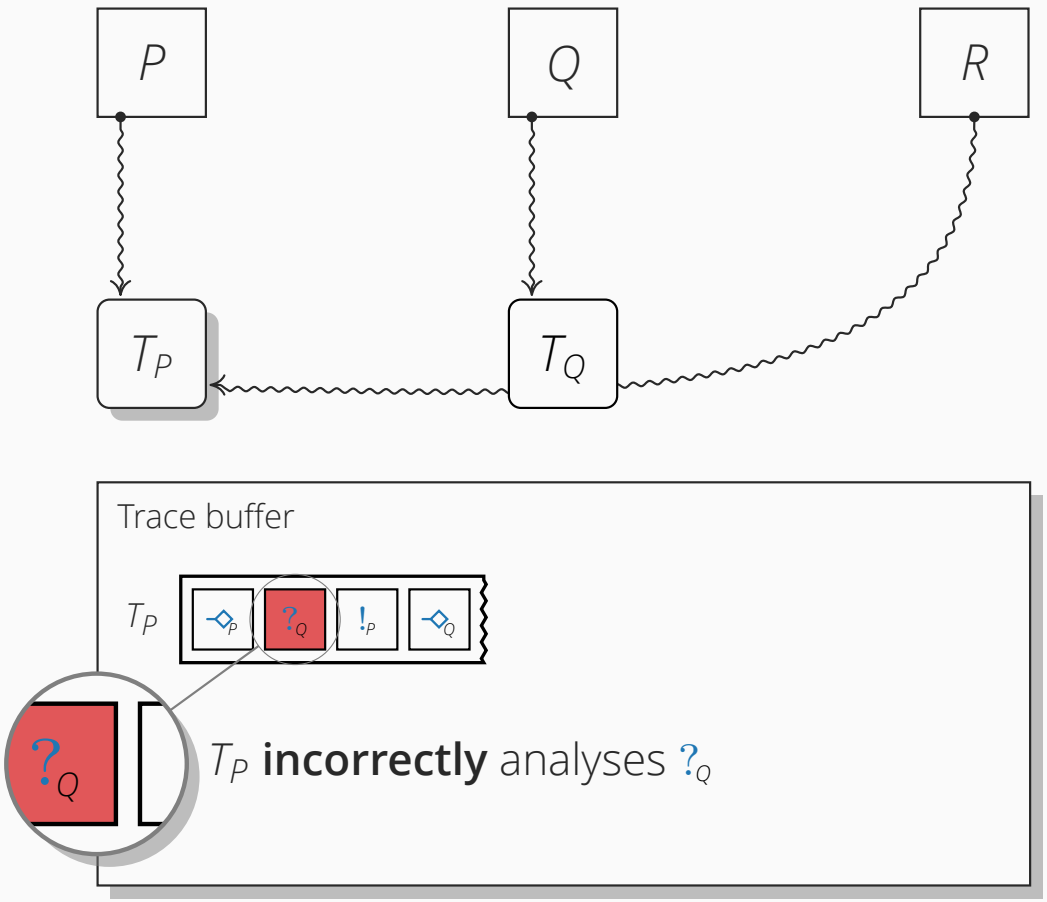
Incomplete traces



The effects of asynchronous tracing

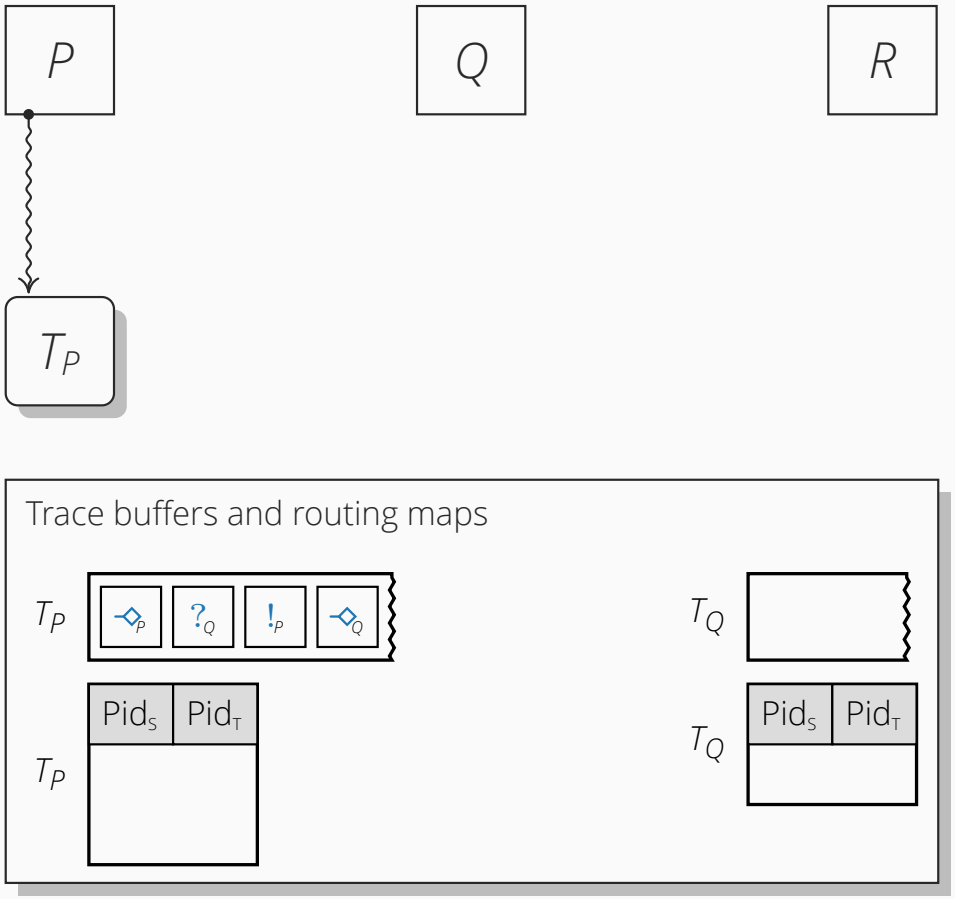
Interleavings between system and tracers

Incomplete traces



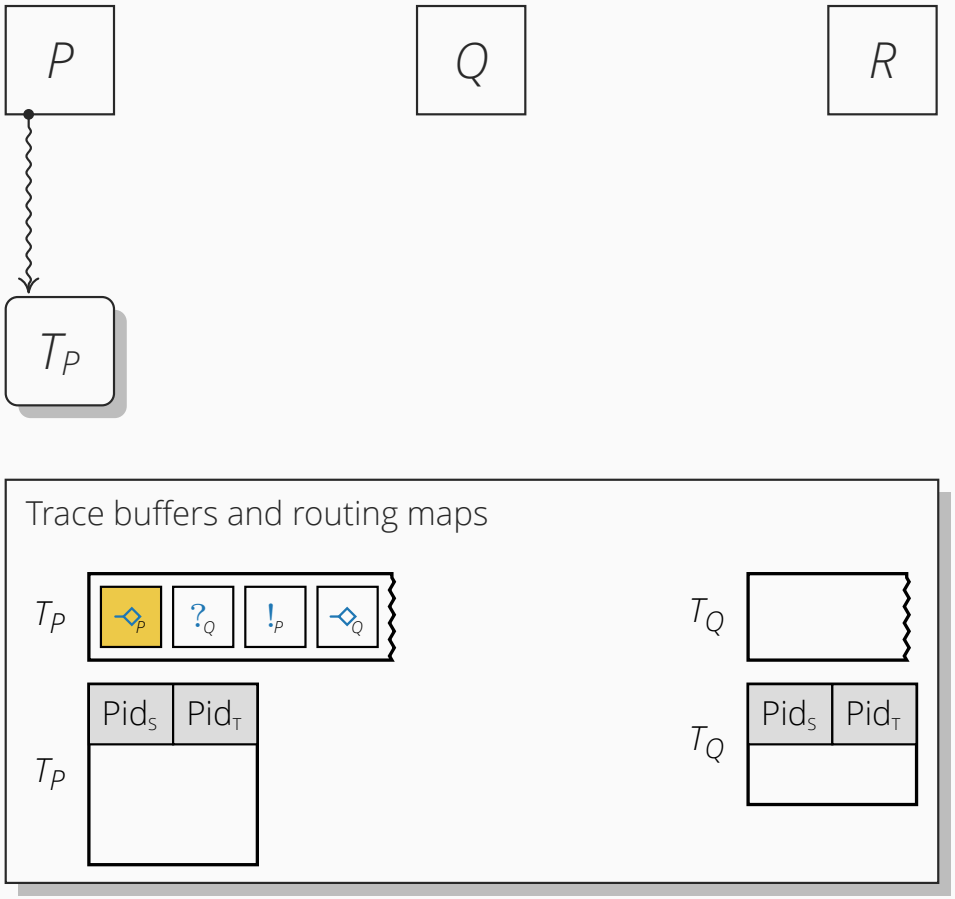
Ensuring complete traces

Tracers forward events to others using **next-hop** routing



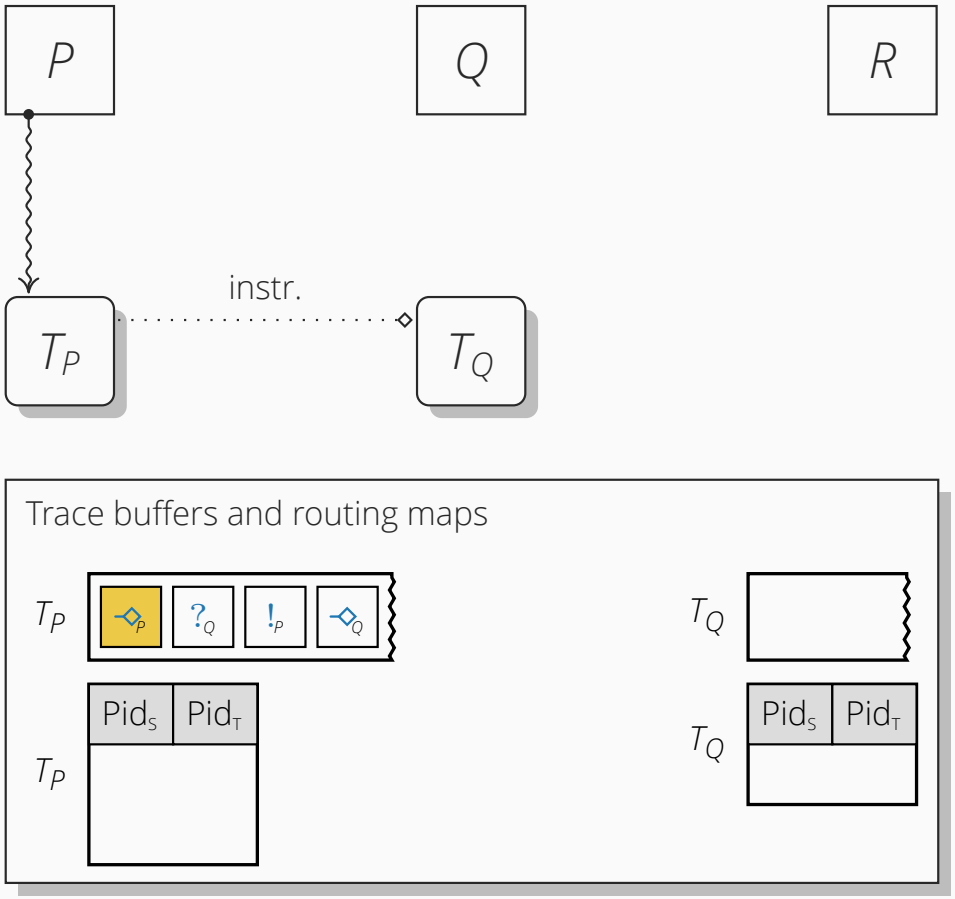
Ensuring complete traces

Tracers forward events to others using **next-hop** routing



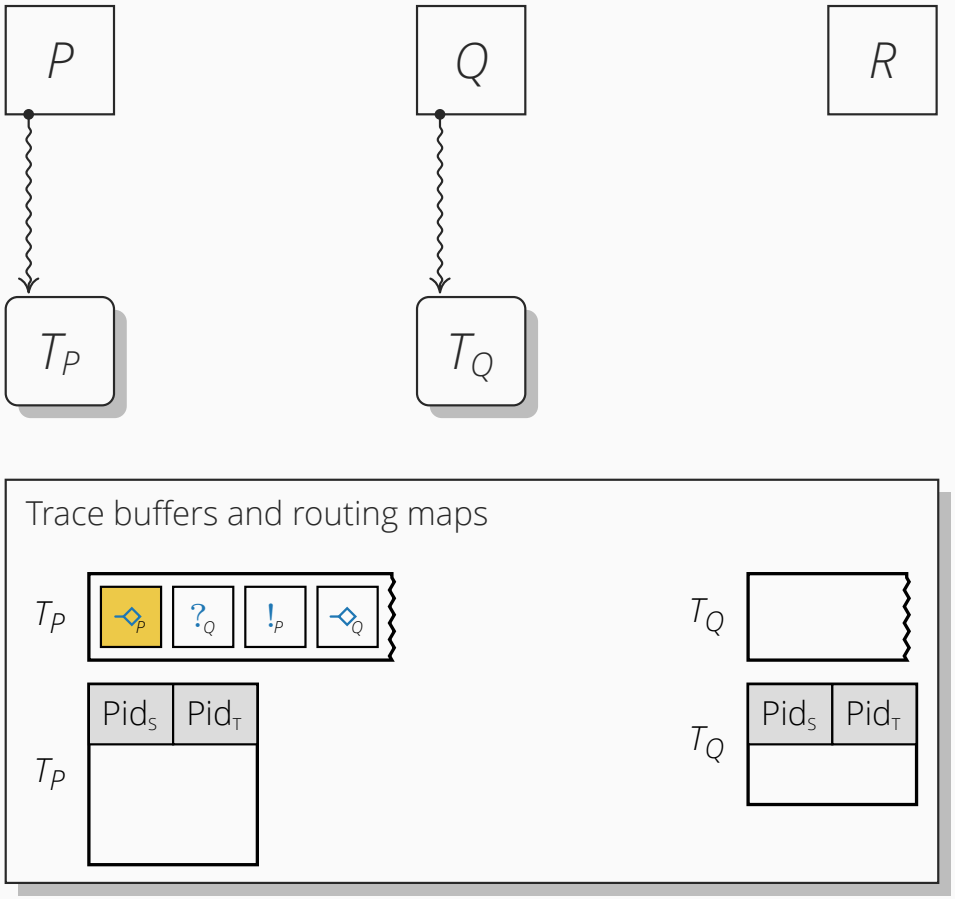
Ensuring complete traces

Tracers forward events to others using **next-hop** routing



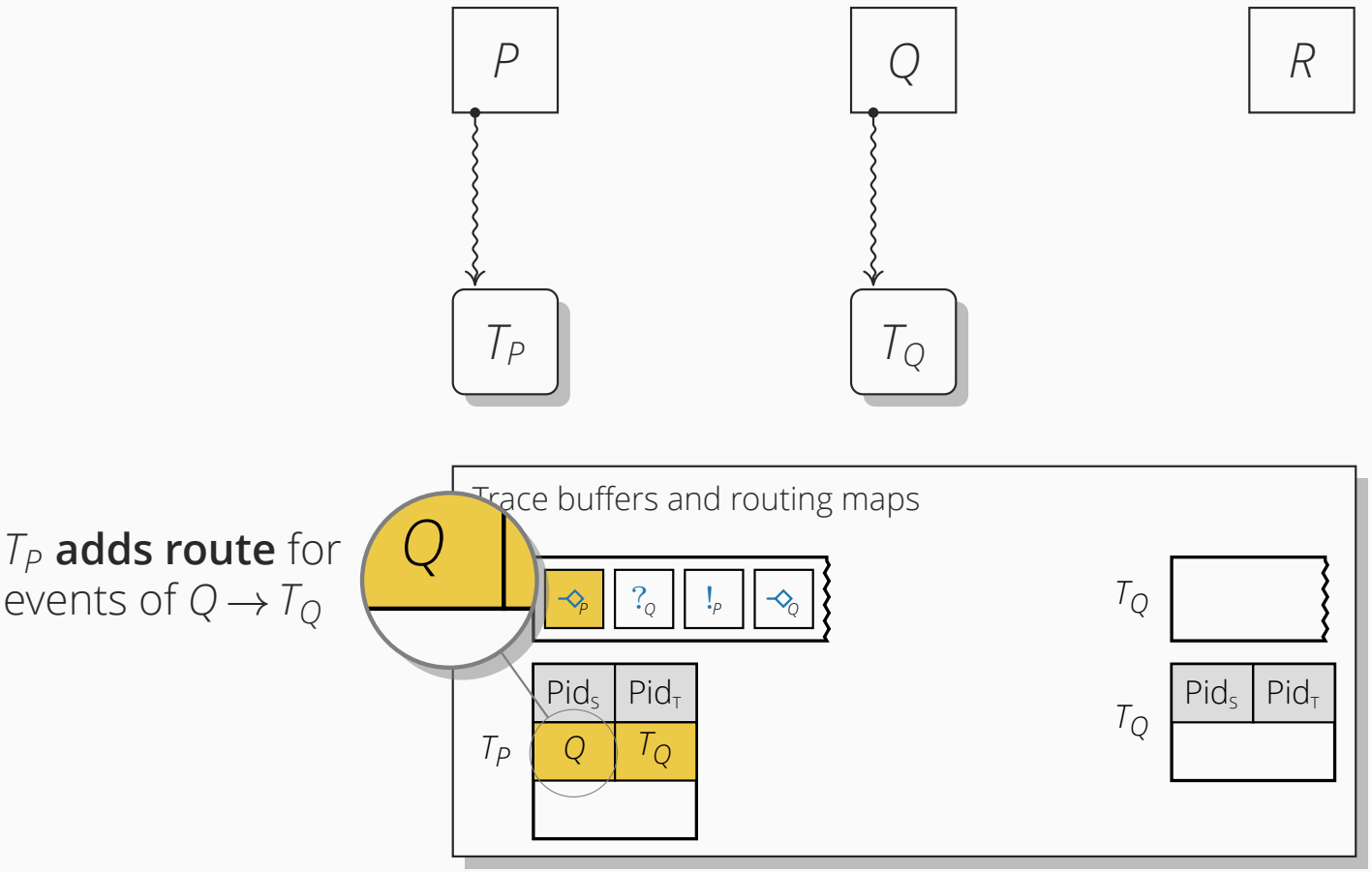
Ensuring complete traces

Tracers forward events to others using **next-hop** routing



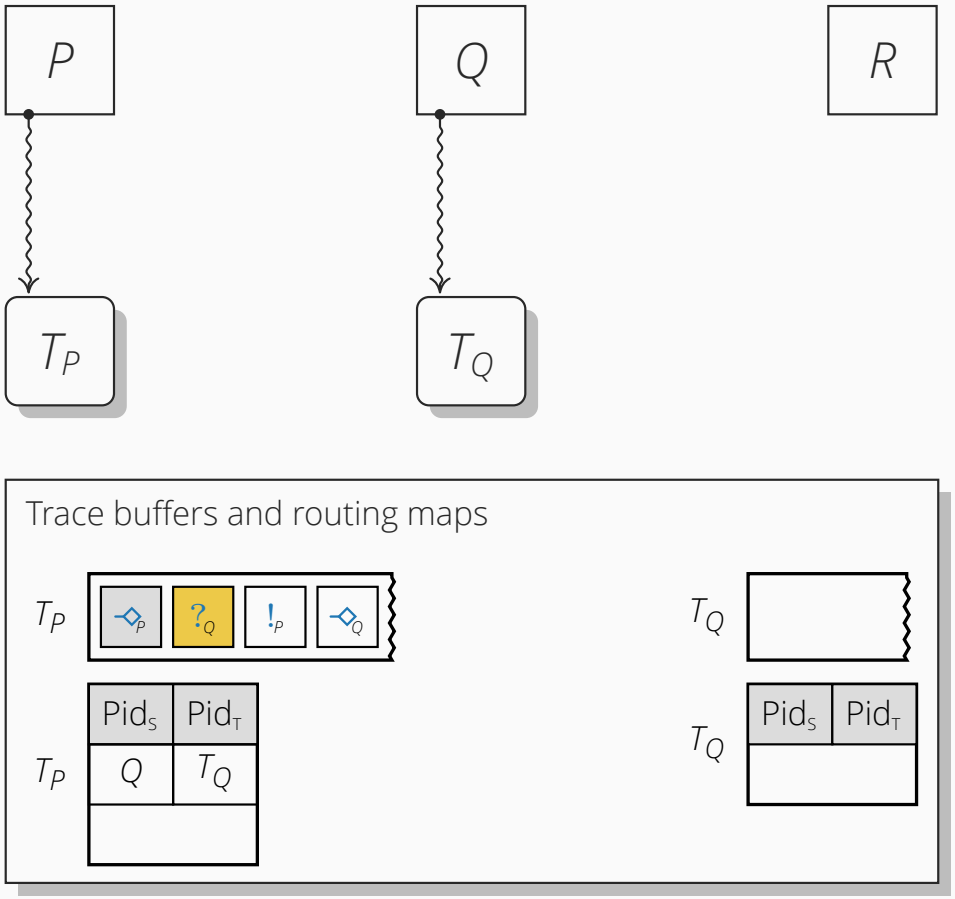
Ensuring complete traces

Tracers forward events to others using **next-hop** routing



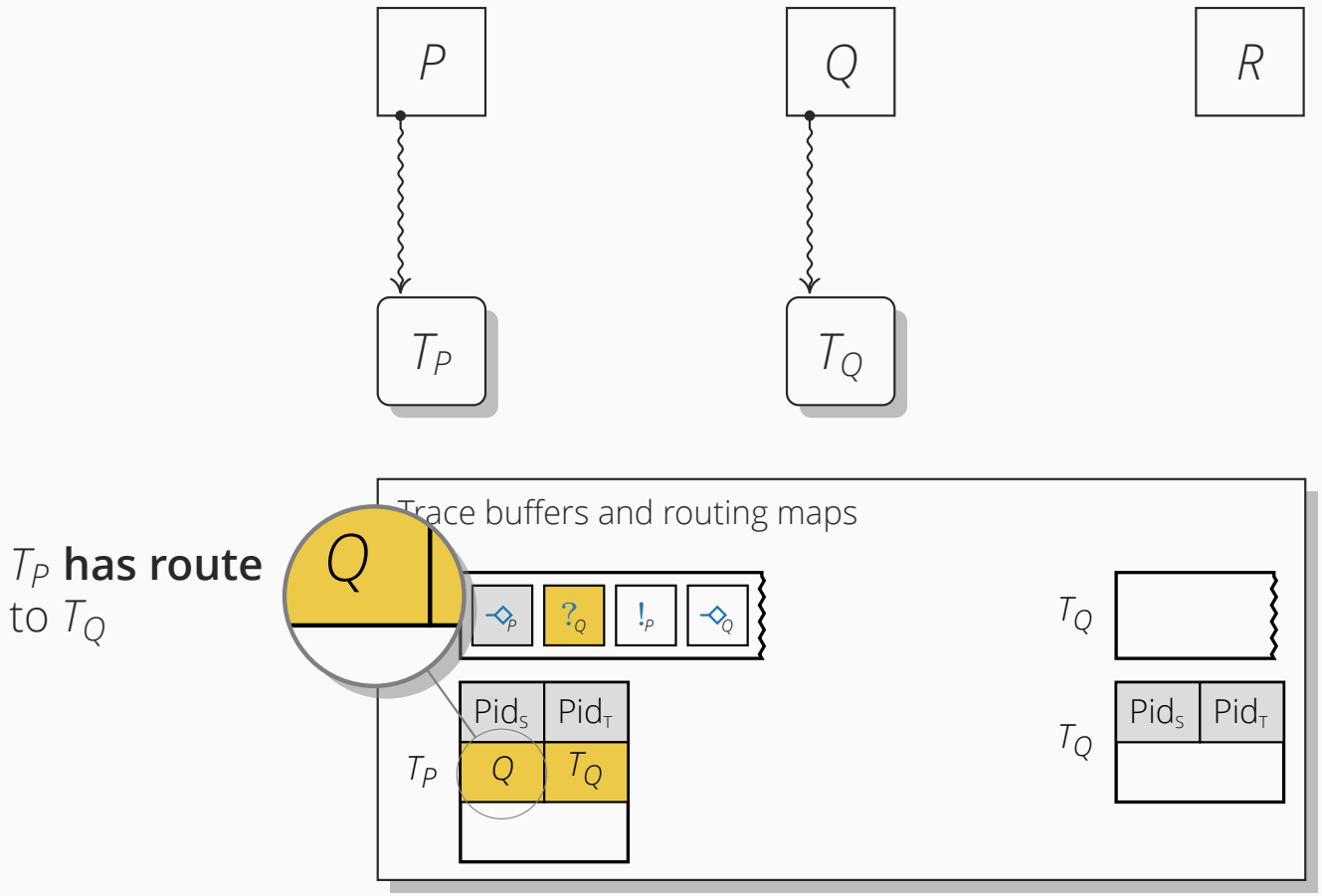
Ensuring complete traces

Tracers forward events to others using **next-hop** routing



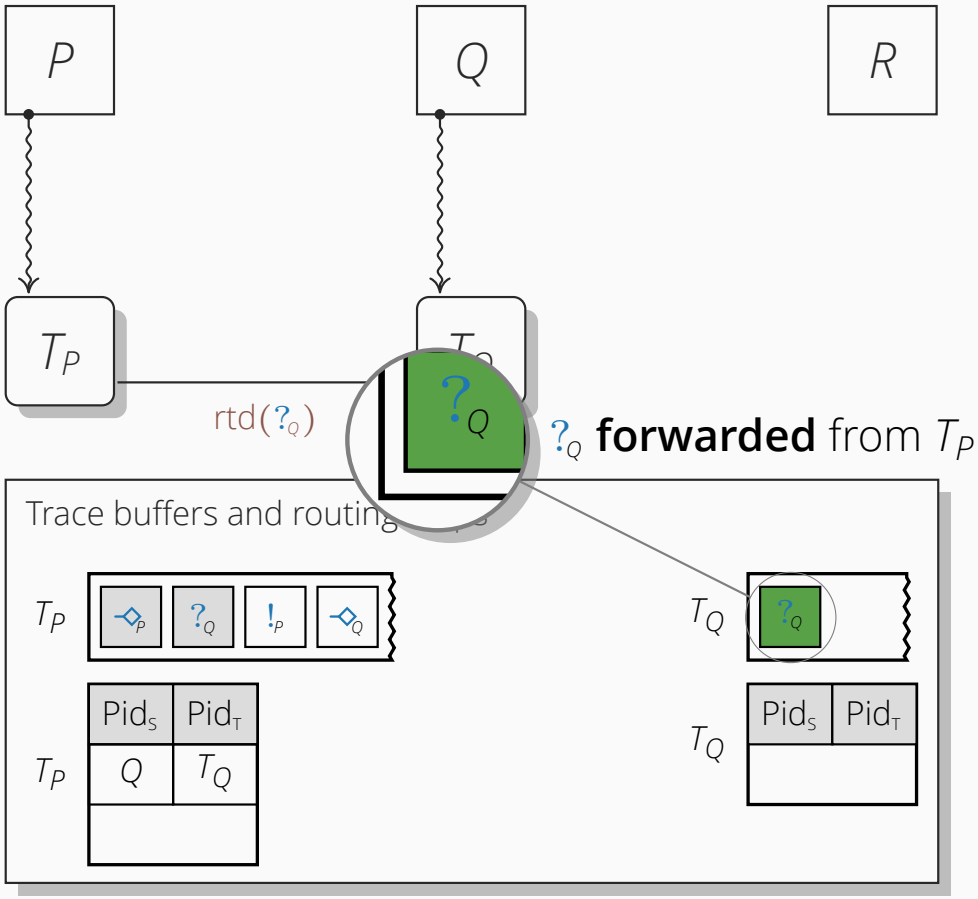
Ensuring complete traces

Tracers forward events to others using **next-hop** routing



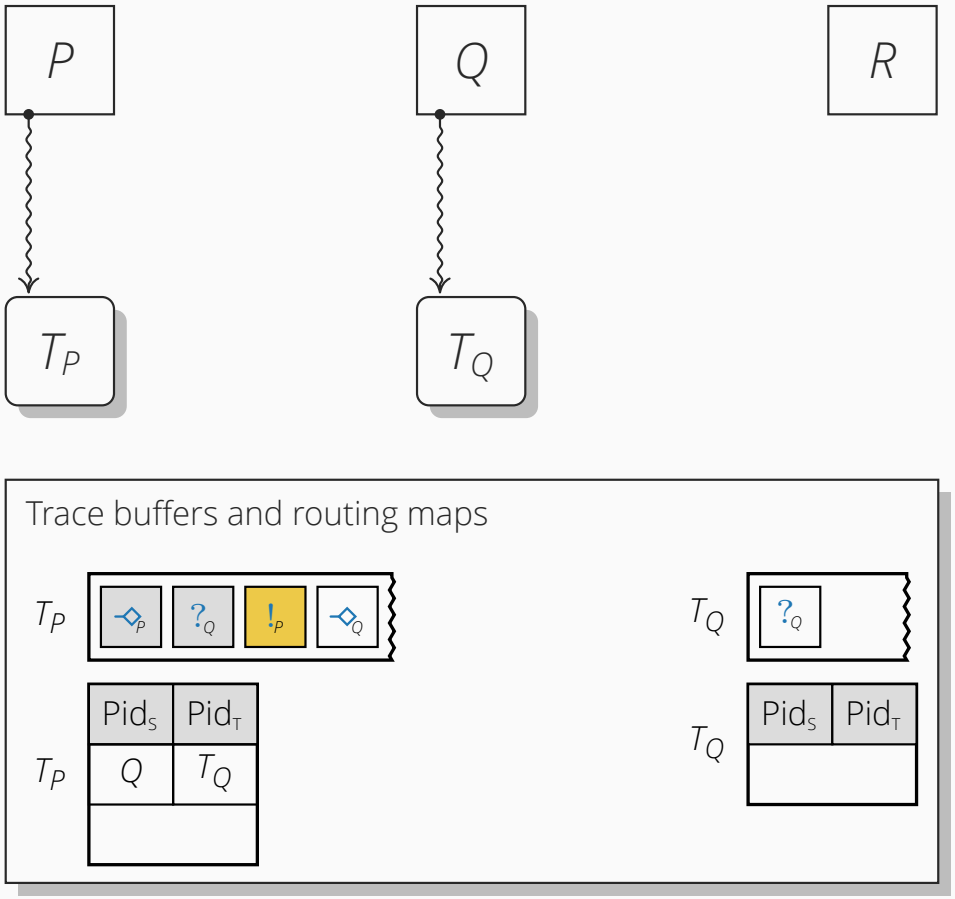
Ensuring complete traces

Tracers forward events to others using **next-hop** routing



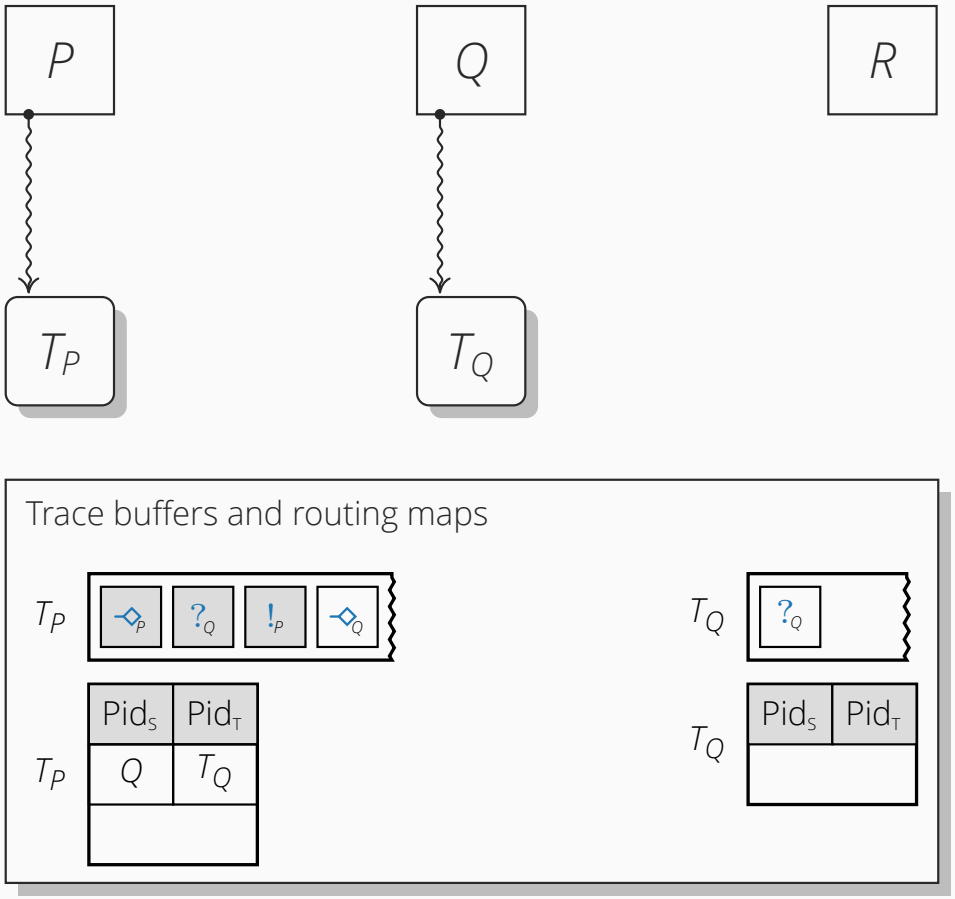
Ensuring complete traces

Tracers forward events to others using **next-hop** routing



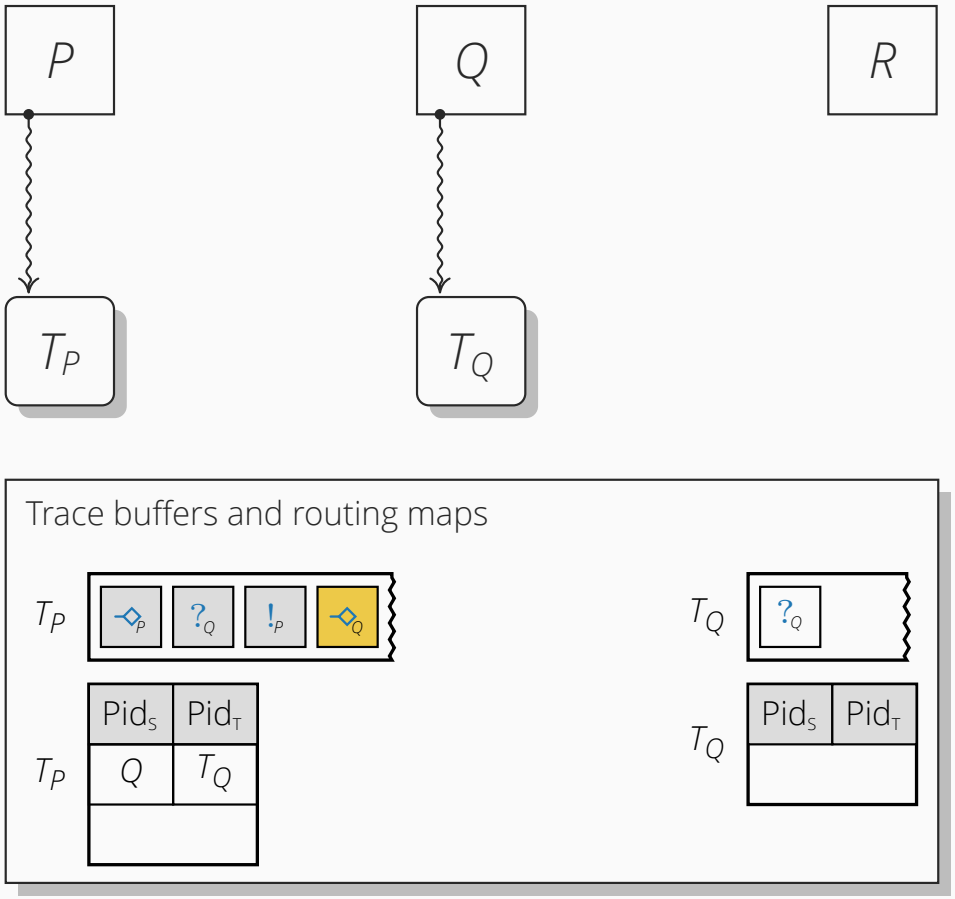
Ensuring complete traces

Tracers forward events to others using **next-hop** routing



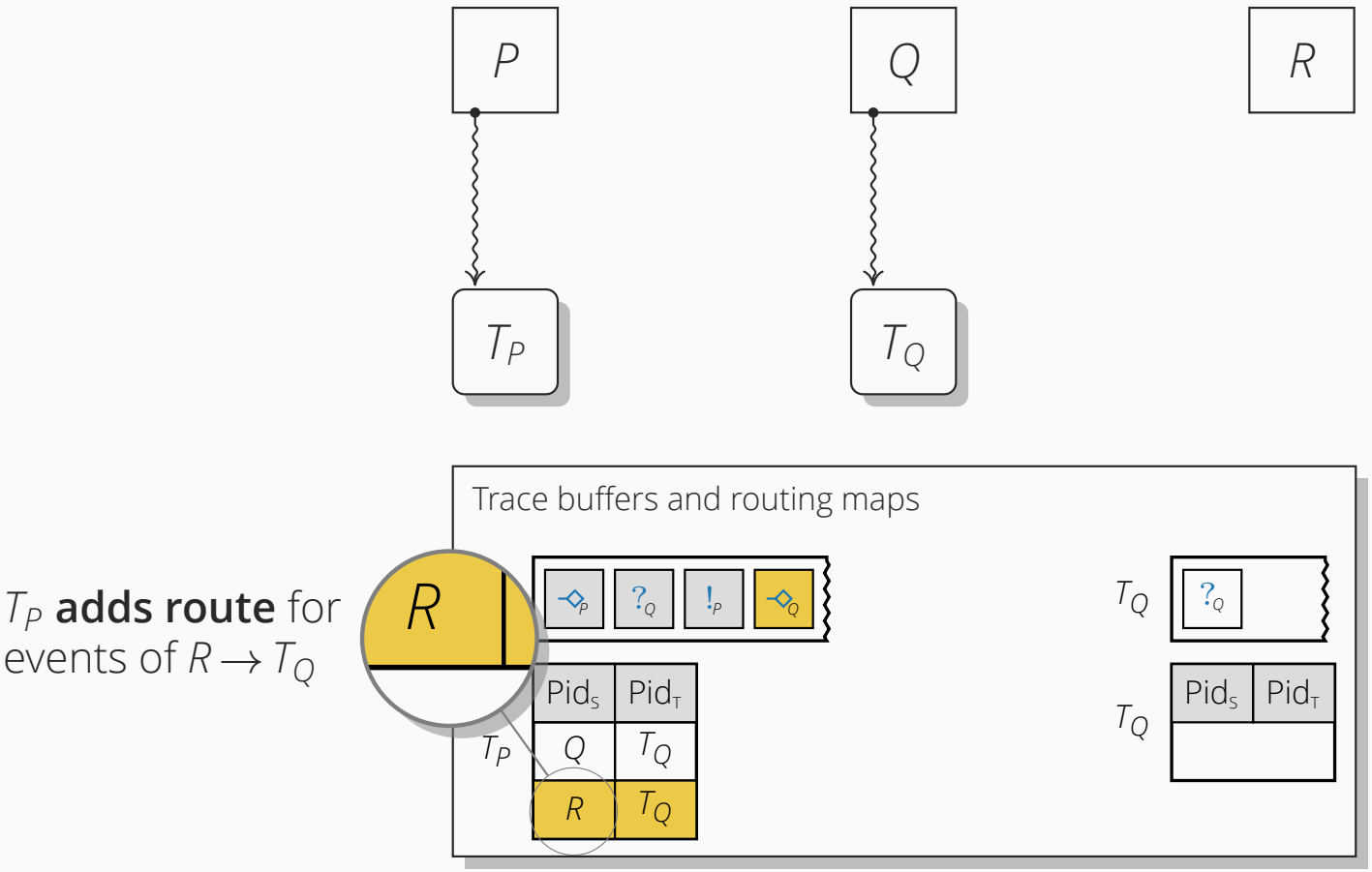
Ensuring complete traces

Tracers forward events to others using **next-hop** routing



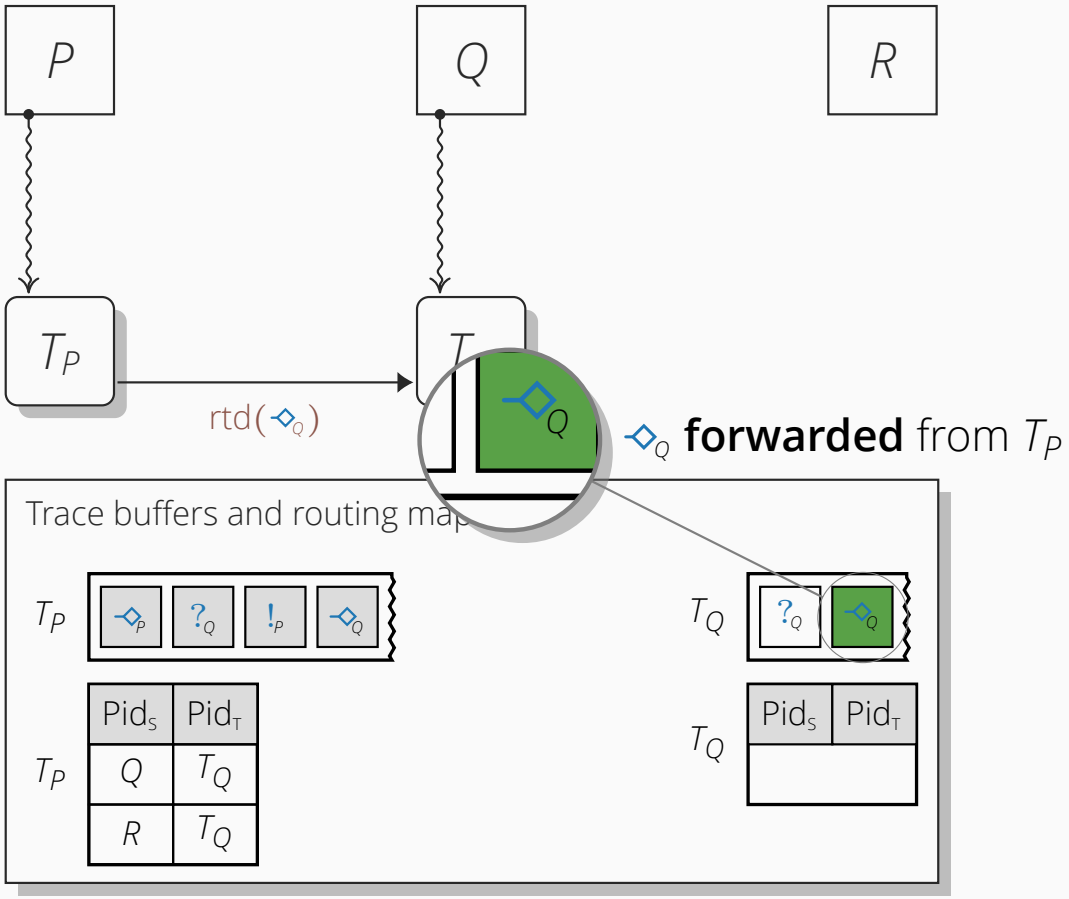
Ensuring complete traces

Tracers forward events to others using **next-hop** routing



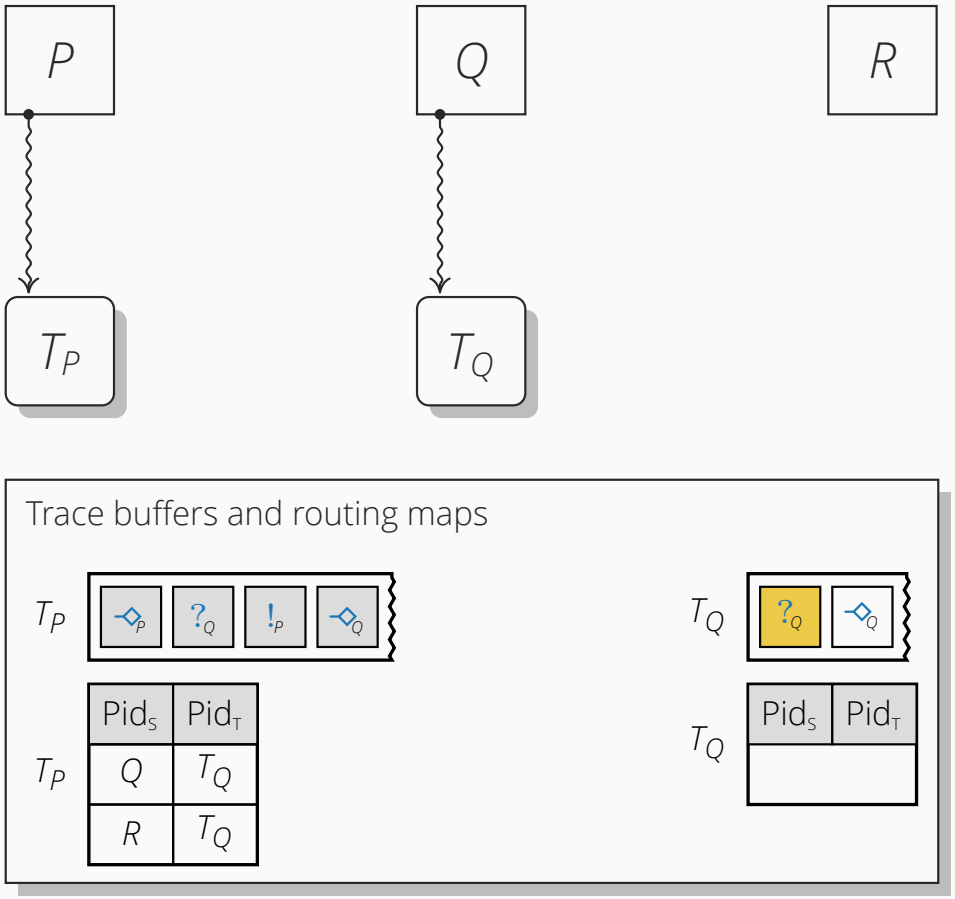
Ensuring complete traces

Tracers forward events to others using **next-hop** routing



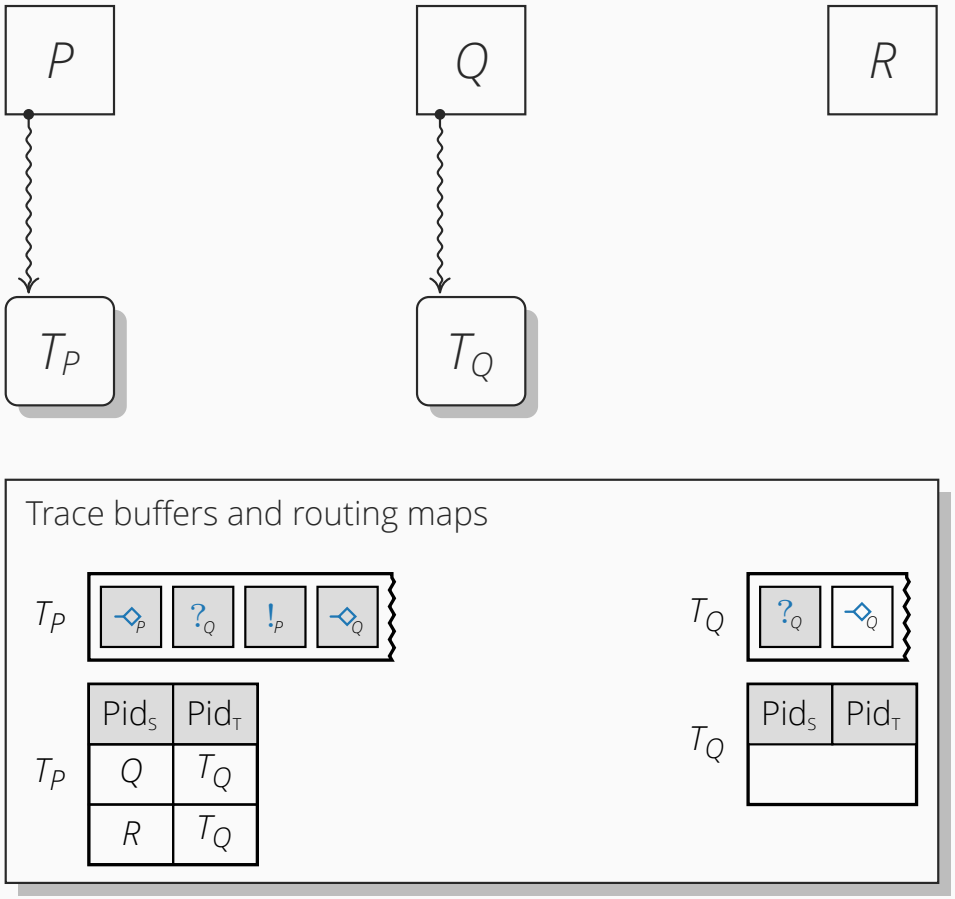
Ensuring complete traces

Tracers forward events to others using **next-hop** routing



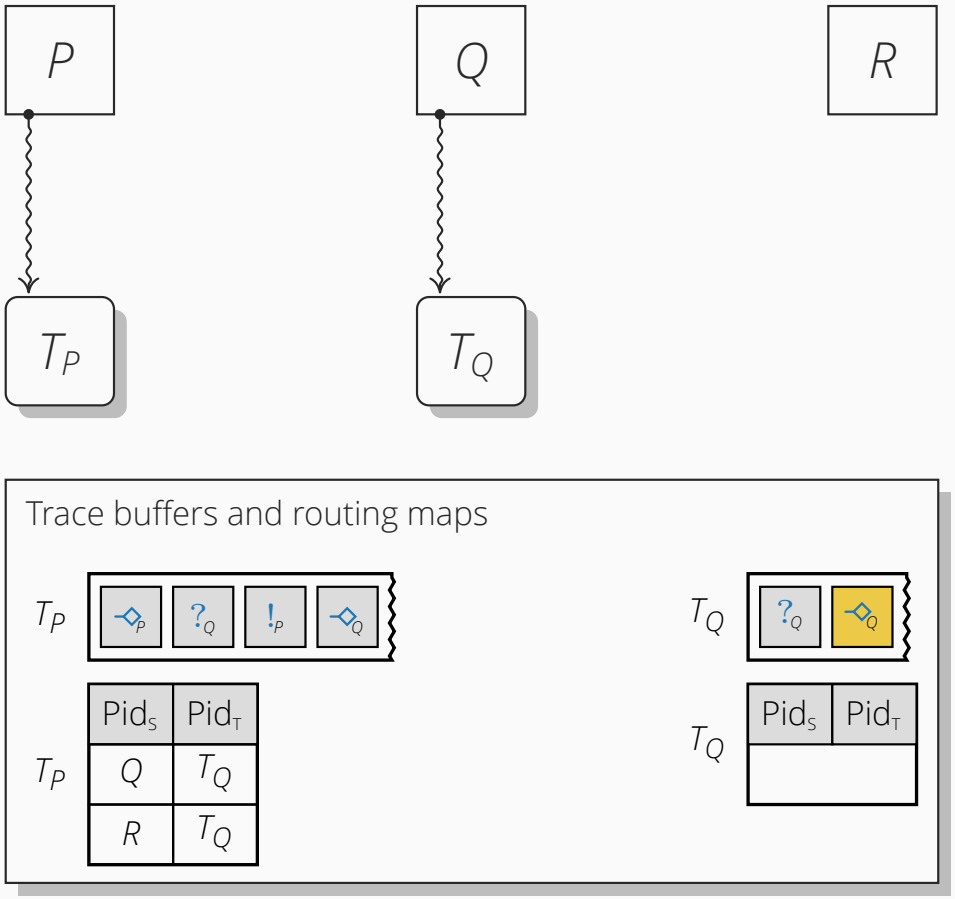
Ensuring complete traces

Tracers forward events to others using **next-hop** routing



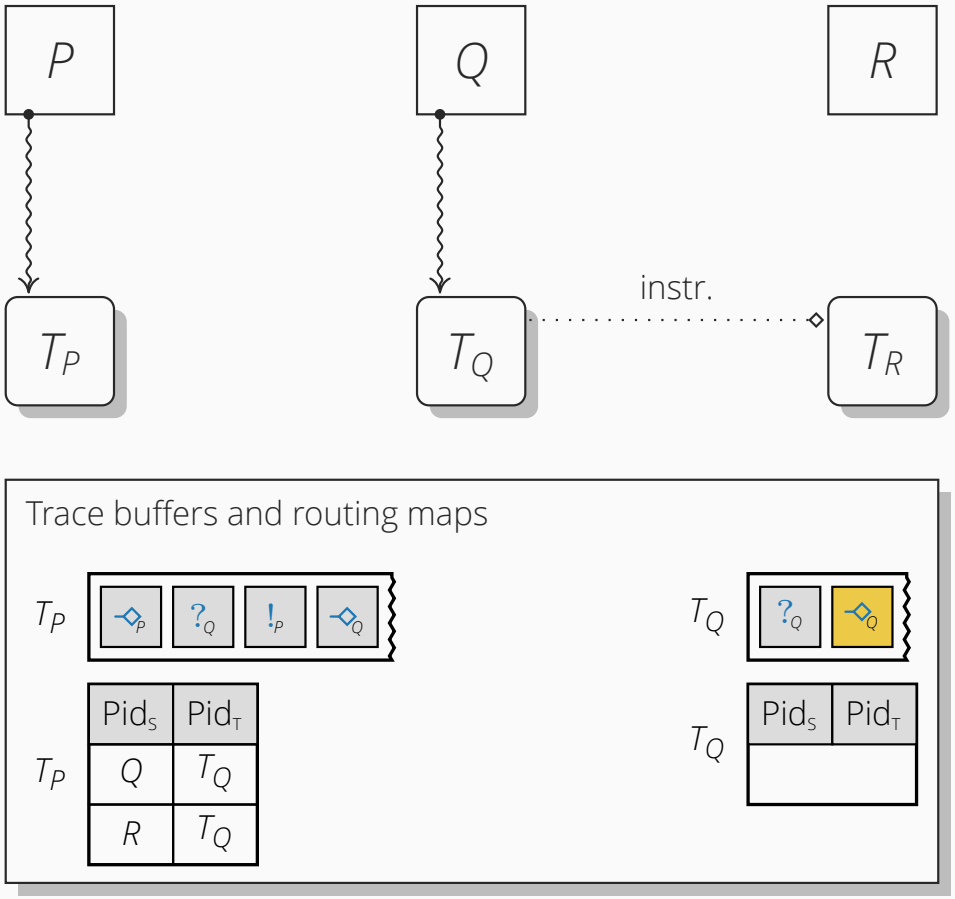
Ensuring complete traces

Tracers forward events to others using **next-hop** routing



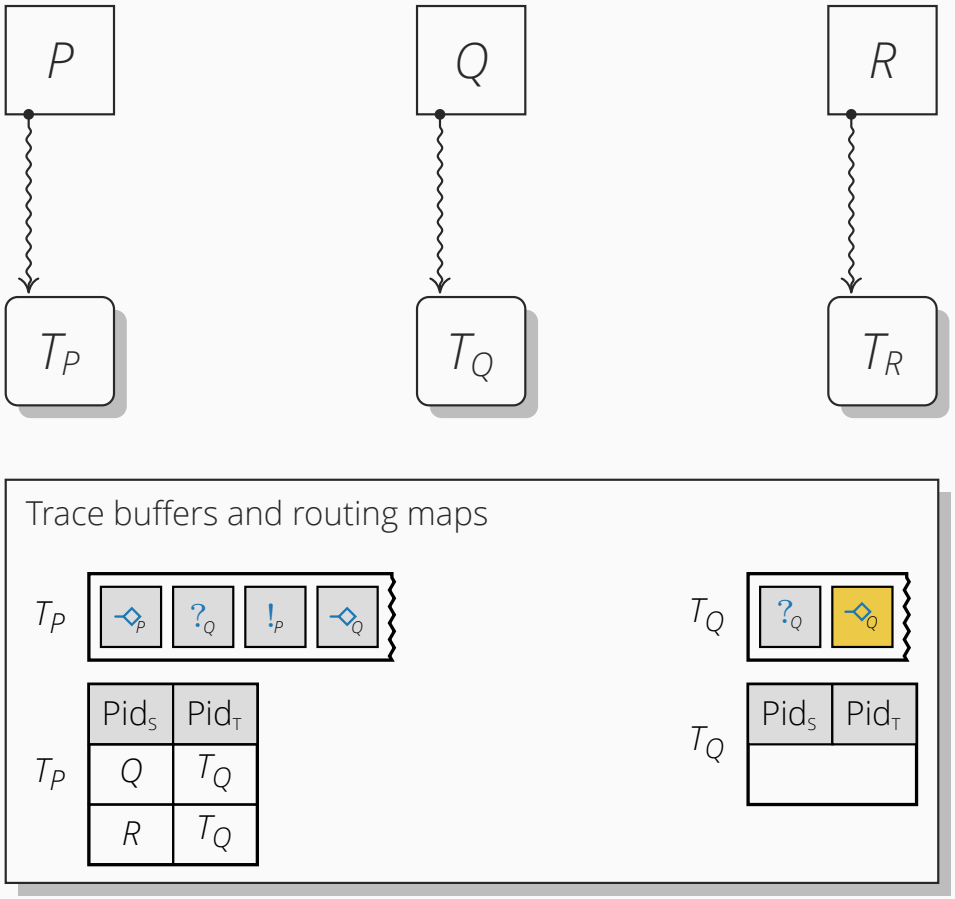
Ensuring complete traces

Tracers forward events to others using **next-hop** routing



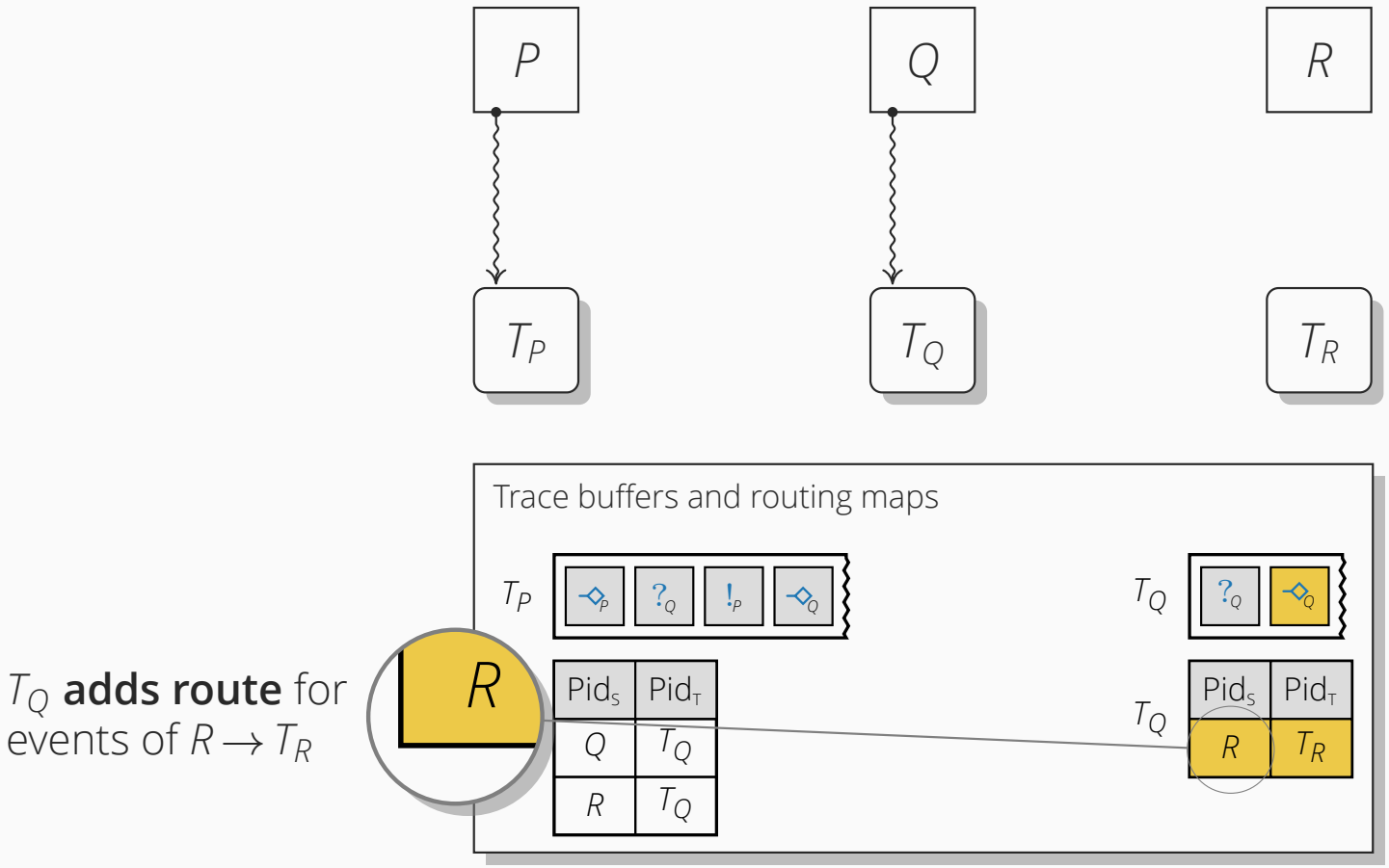
Ensuring complete traces

Tracers forward events to others using **next-hop** routing



Ensuring complete traces

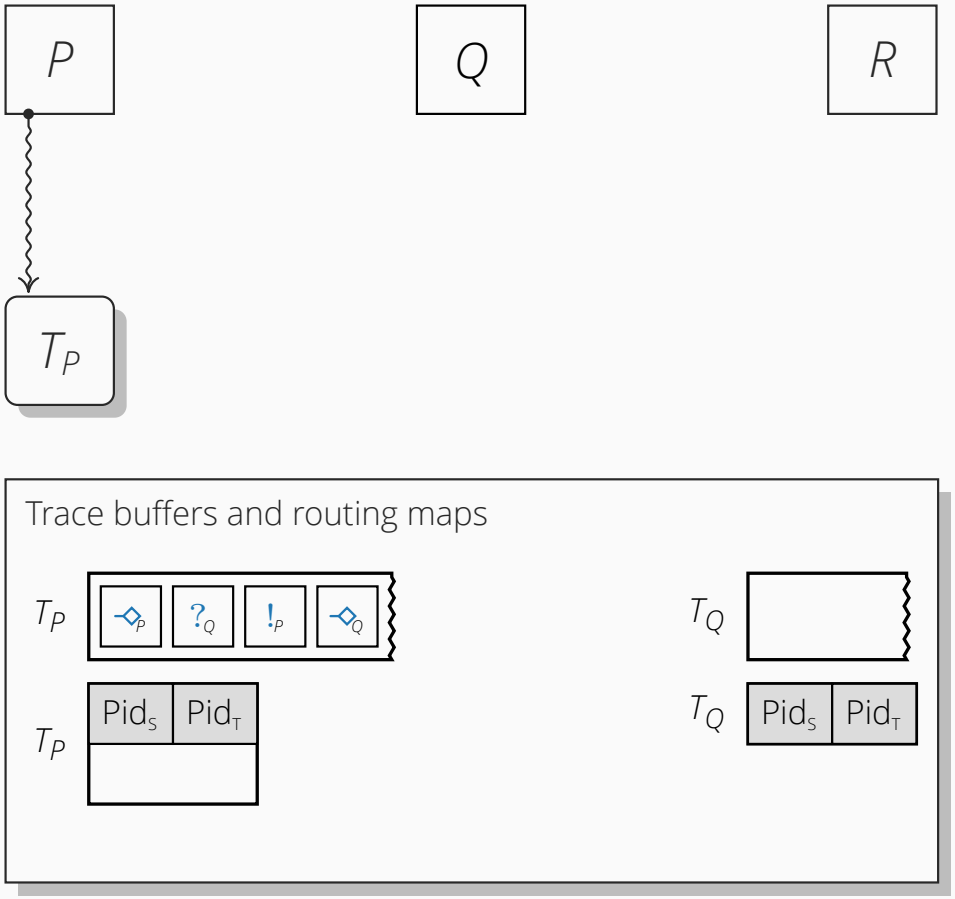
Tracers forward events to others using **next-hop** routing



Next-hop routing is not enough

Tracers can still process events **out of order**

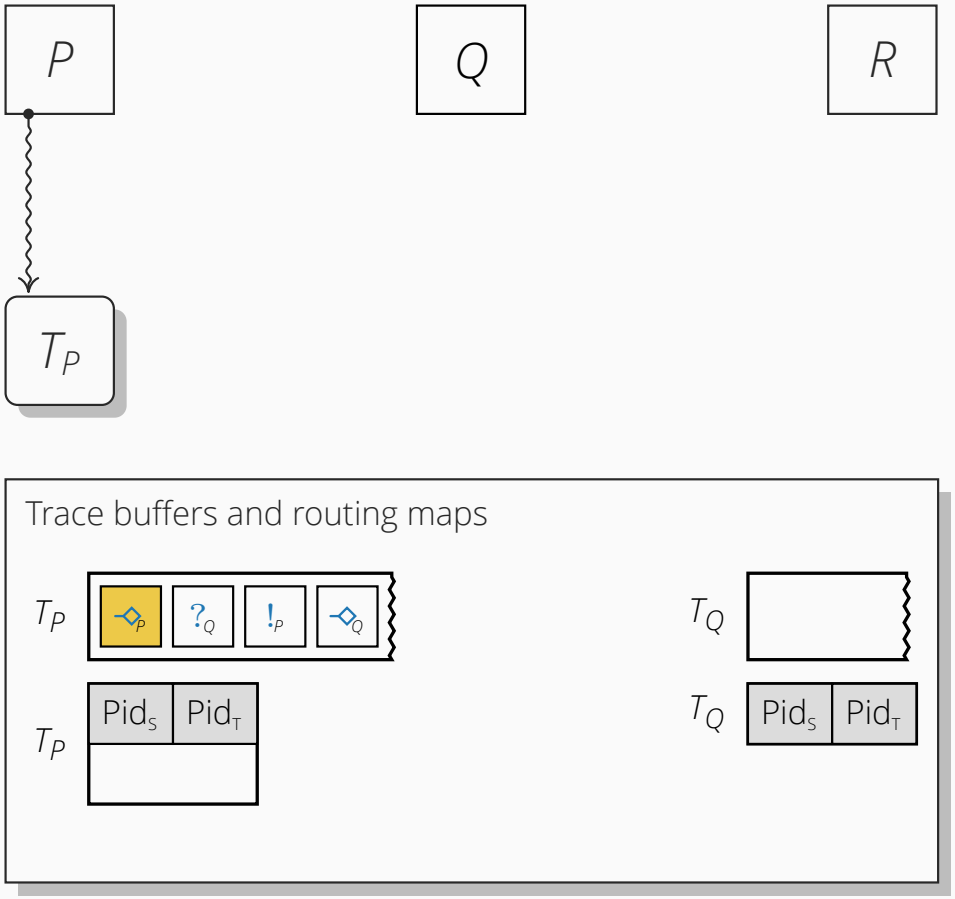
Inconsistent traces



Next-hop routing is not enough

Tracers can still process events **out of order**

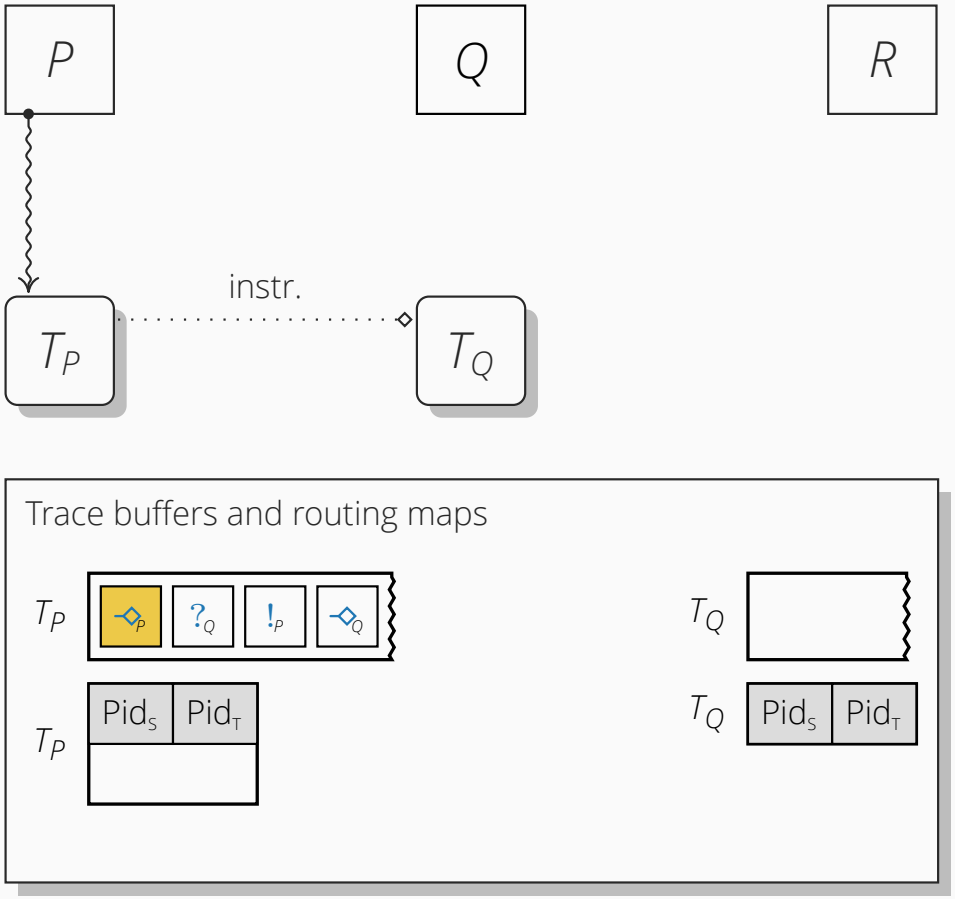
Inconsistent traces



Next-hop routing is not enough

Tracers can still process events **out of order**

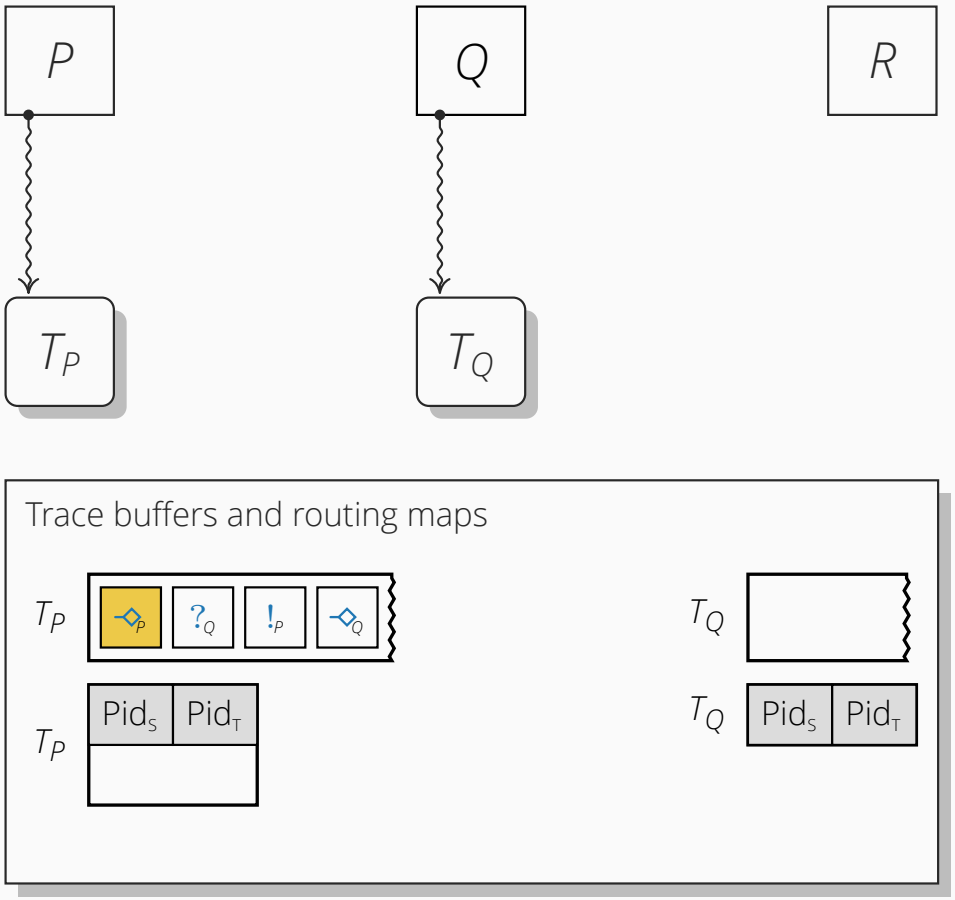
Inconsistent traces



Next-hop routing is not enough

Tracers can still process events **out of order**

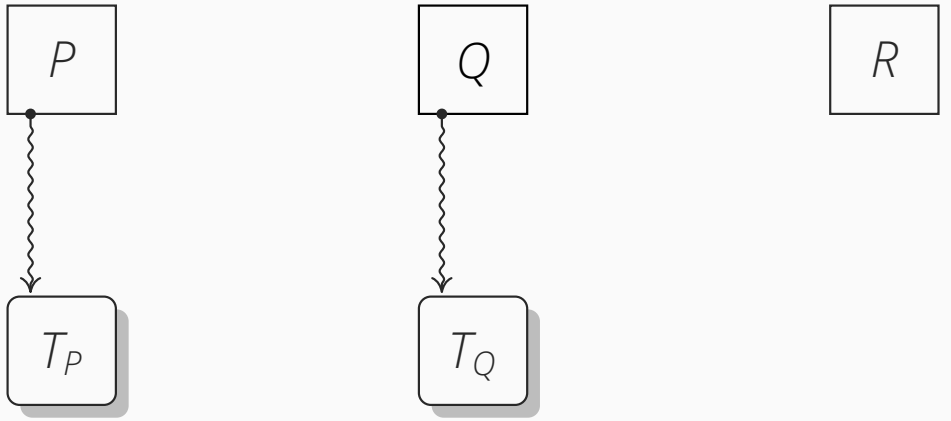
Inconsistent traces



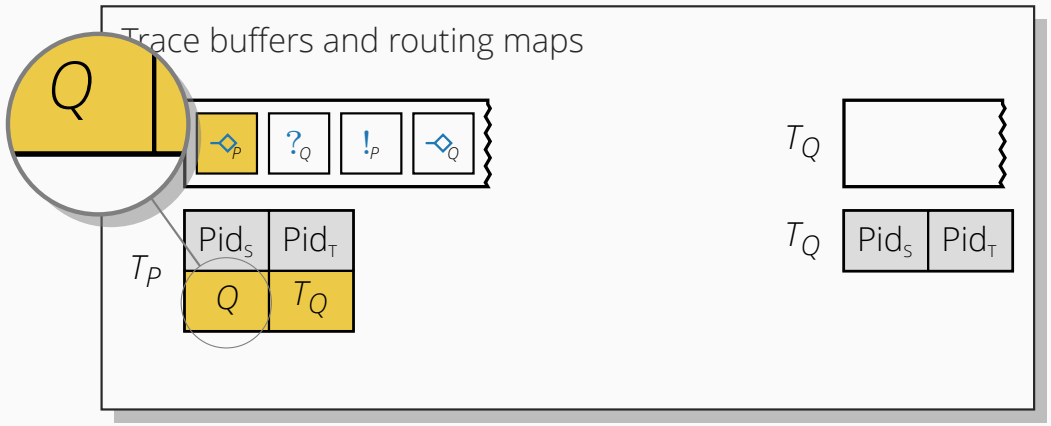
Next-hop routing is not enough

Tracers can still process events **out of order**

Inconsistent traces



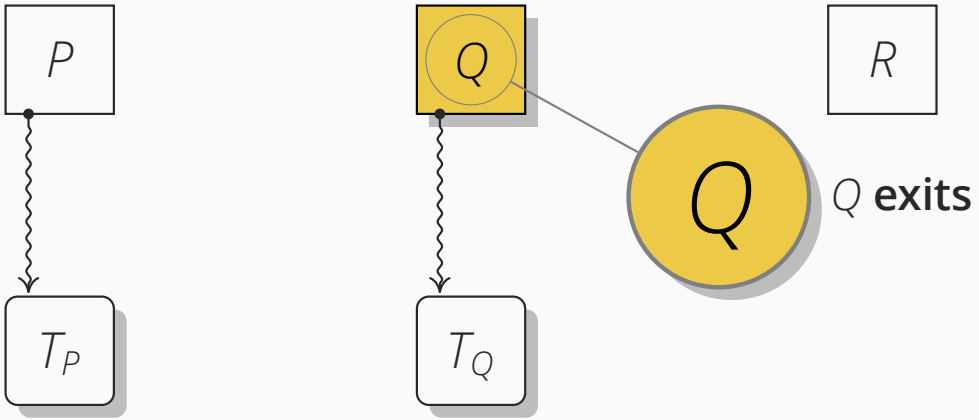
T_P **adds route** for events of $Q \rightarrow T_Q$

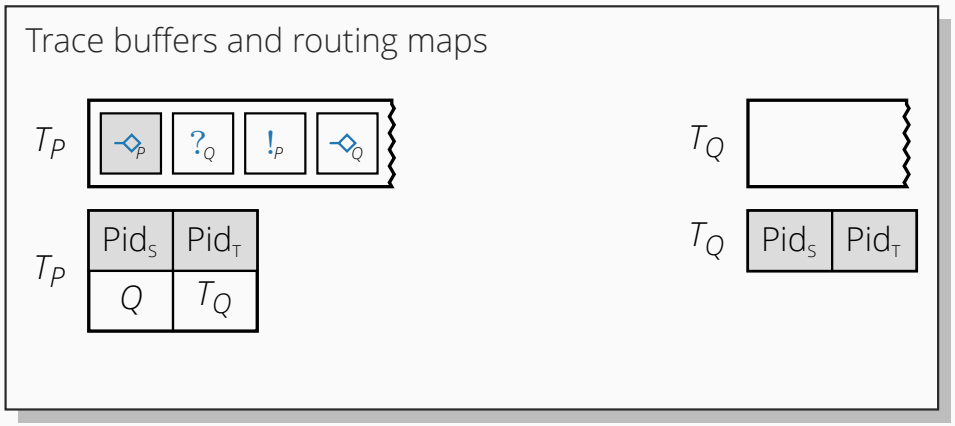


Next-hop routing is not enough

Tracers can still process events **out of order**

Inconsistent traces

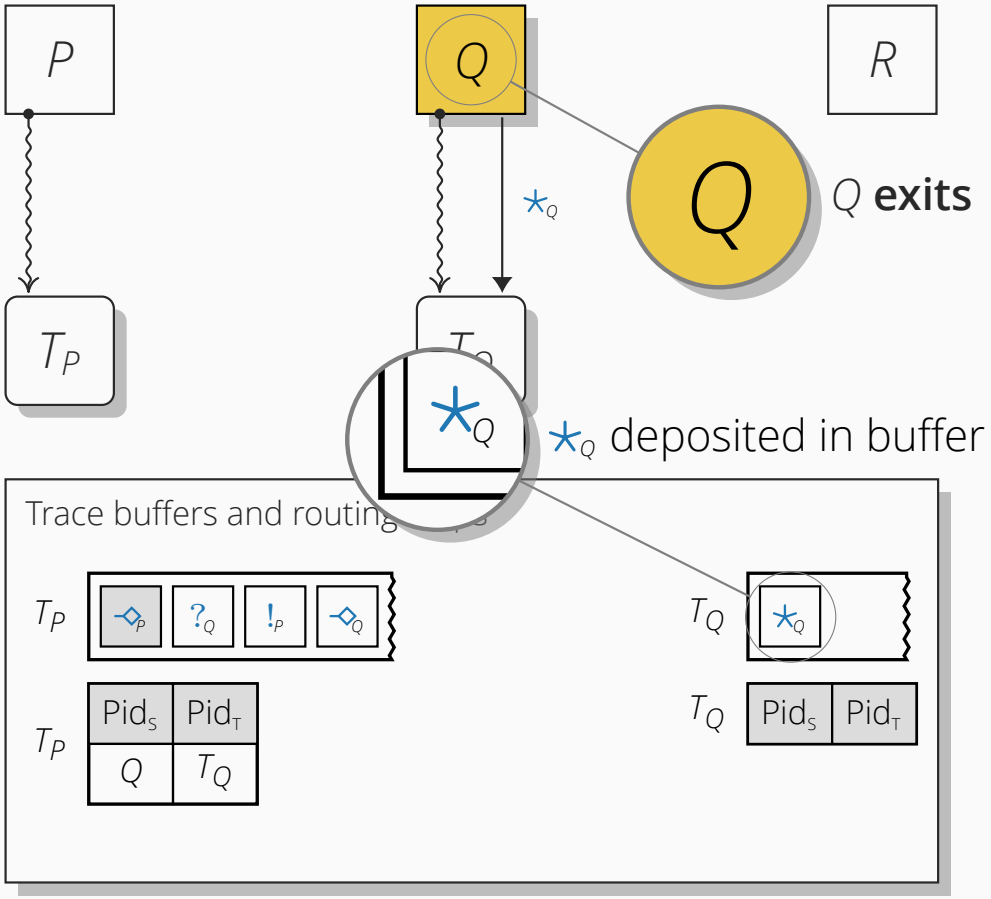




Next-hop routing is not enough

Tracers can still process events **out of order**

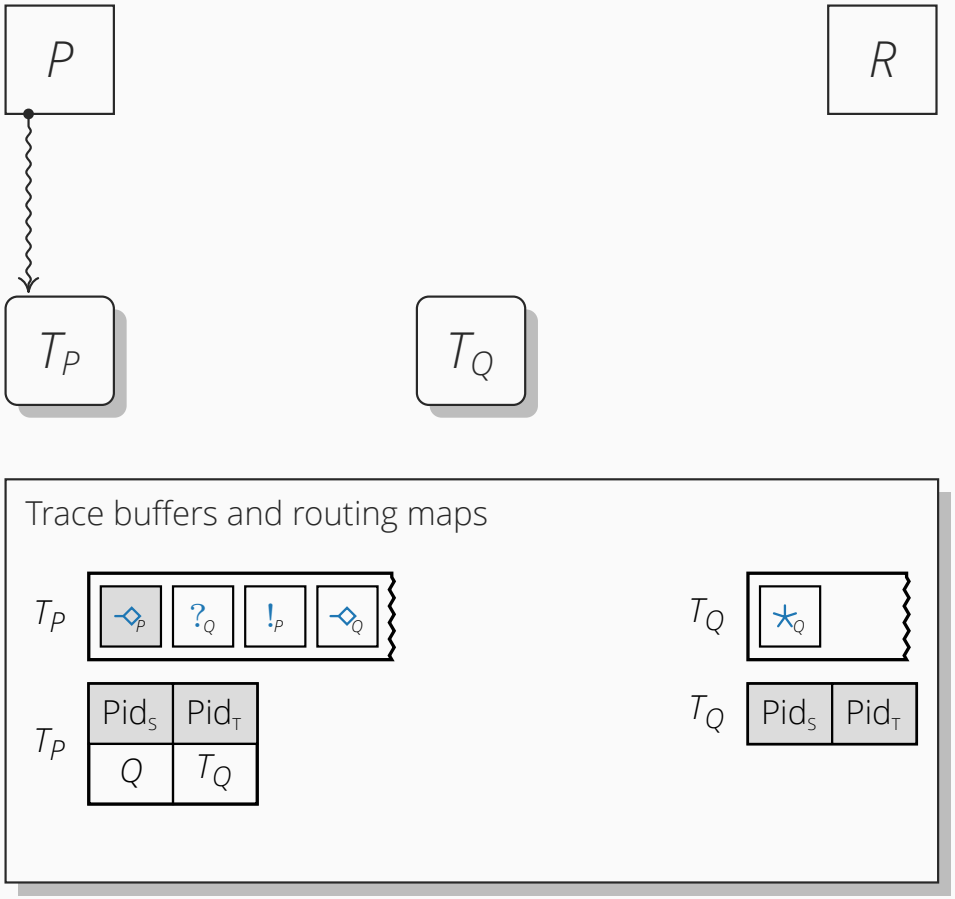
Inconsistent traces



Next-hop routing is not enough

Tracers can still process events **out of order**

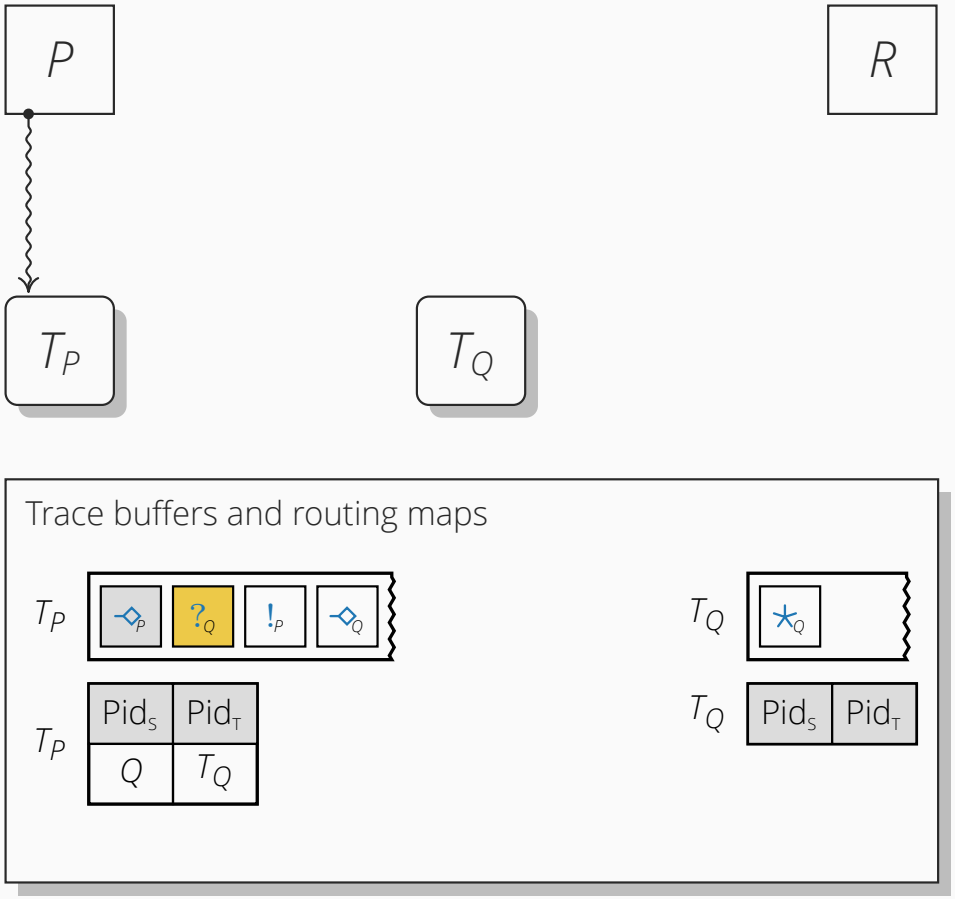
Inconsistent traces



Next-hop routing is not enough

Tracers can still process events **out of order**

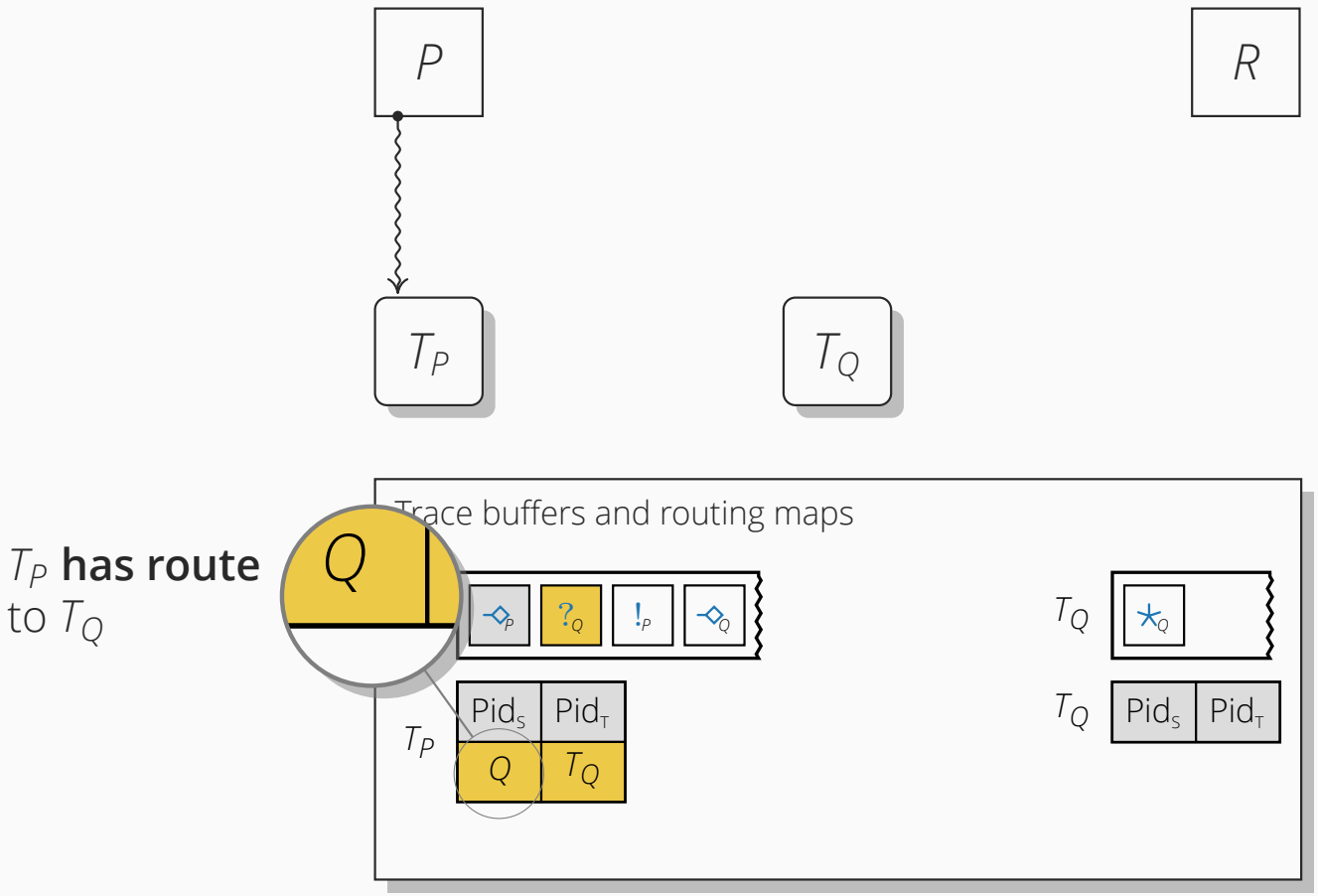
Inconsistent traces



Next-hop routing is not enough

Tracers can still process events **out of order**

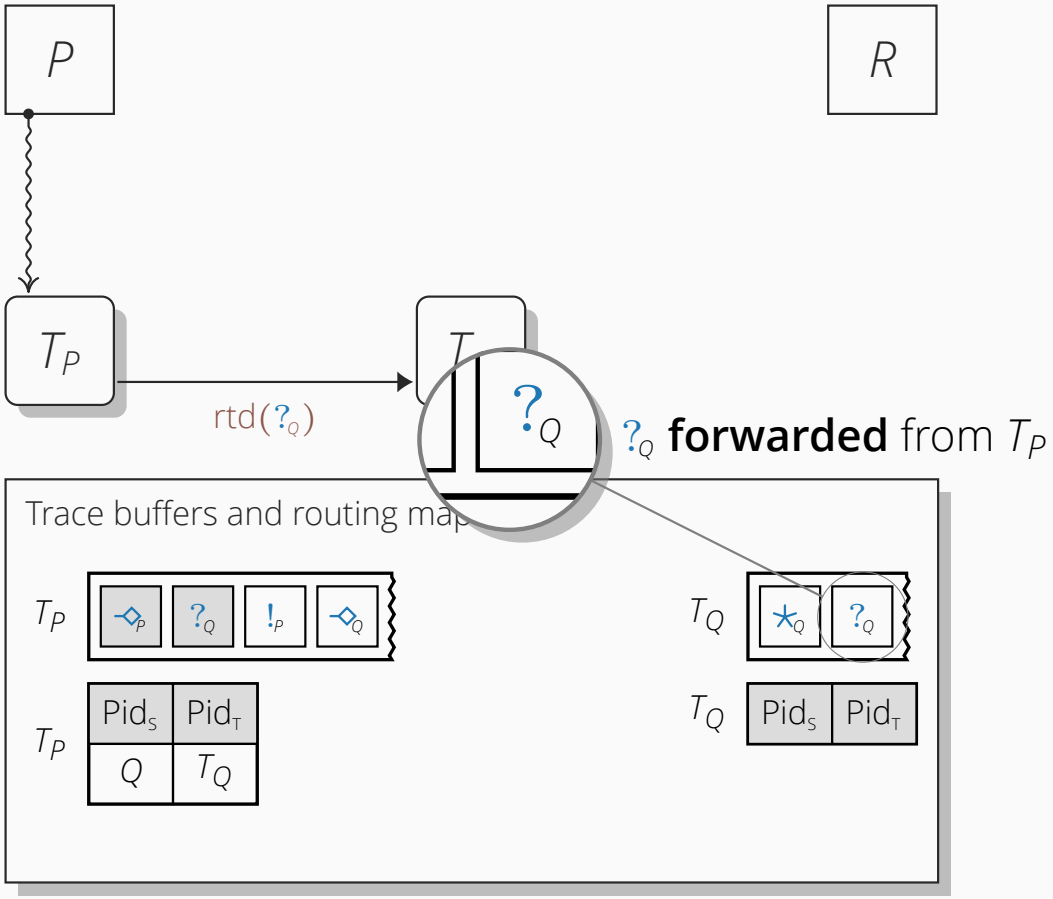
Inconsistent traces



Next-hop routing is not enough

Tracers can still process events **out of order**

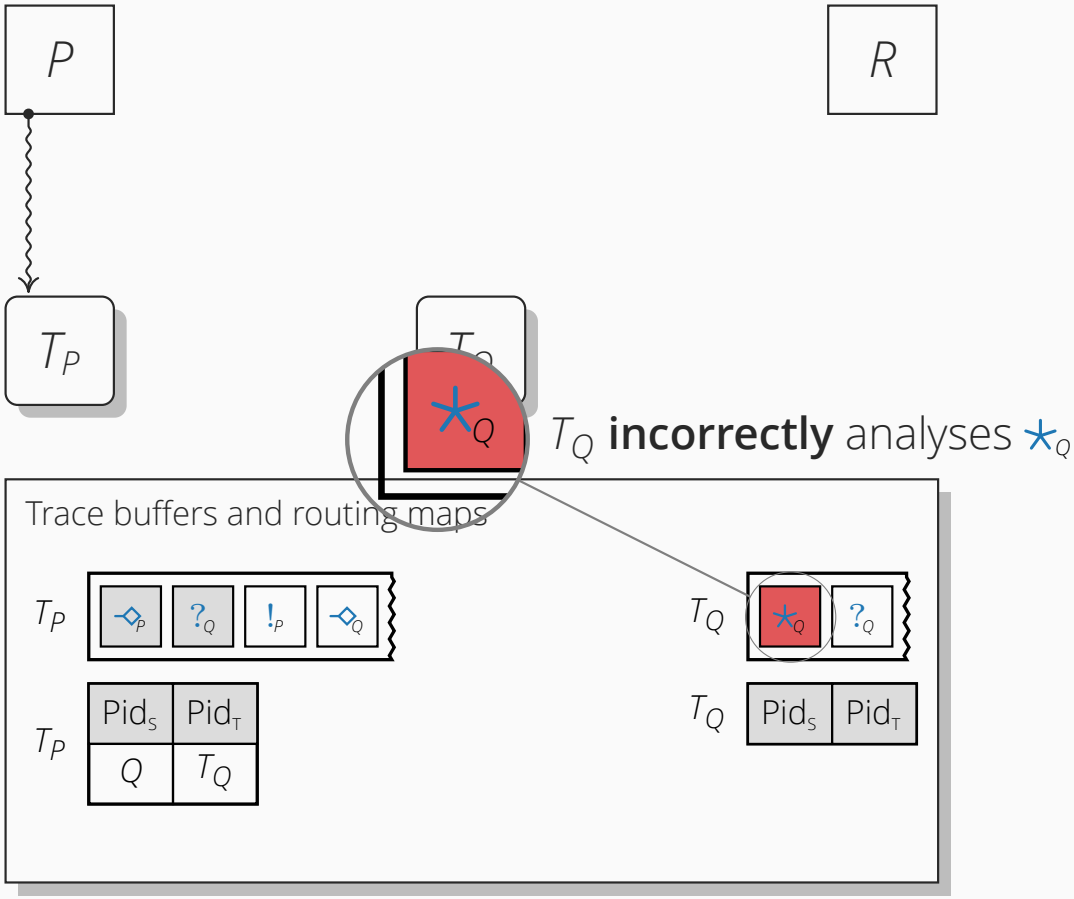
Inconsistent traces



Next-hop routing is not enough

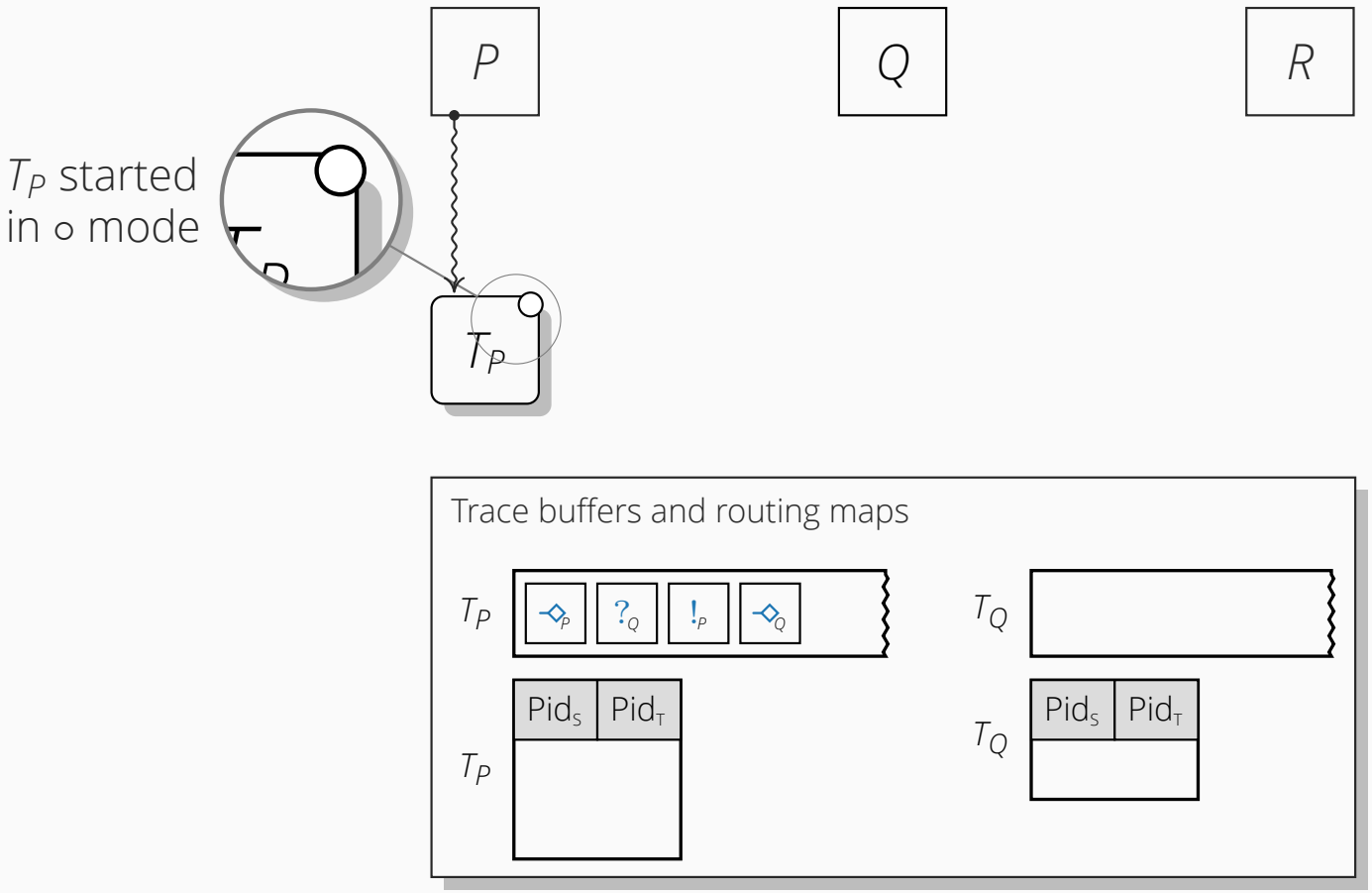
Tracers can still process events **out of order**

Inconsistent traces



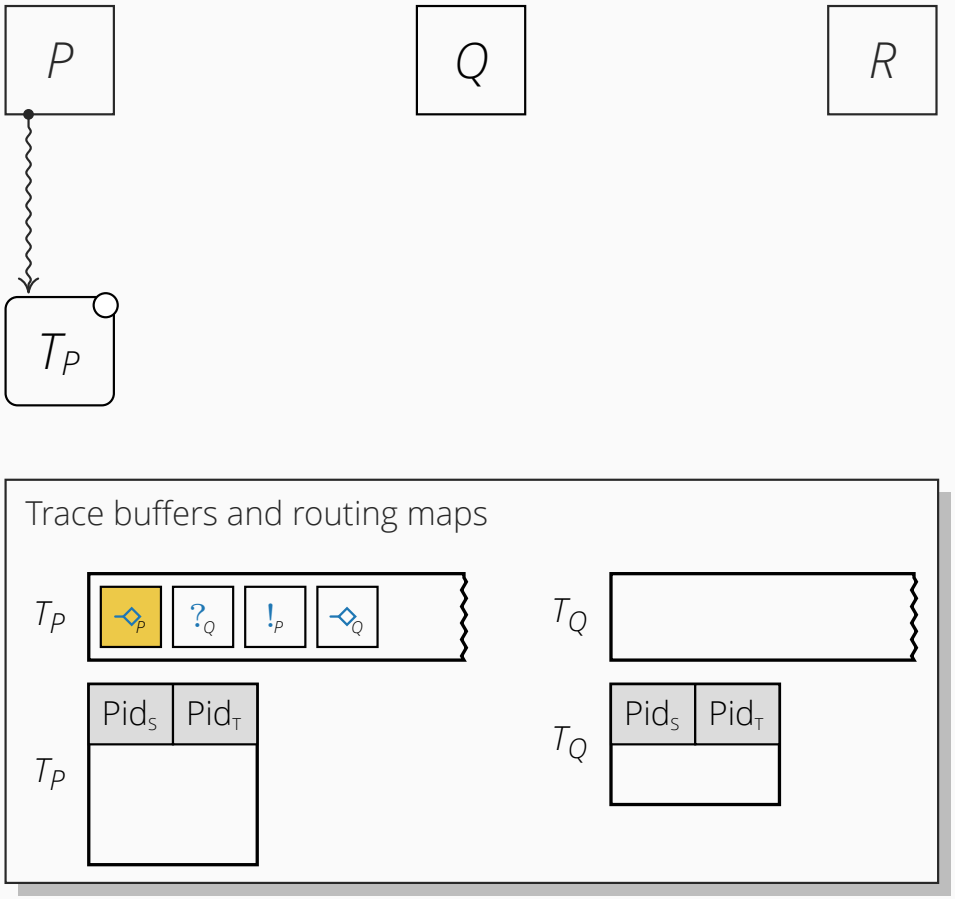
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



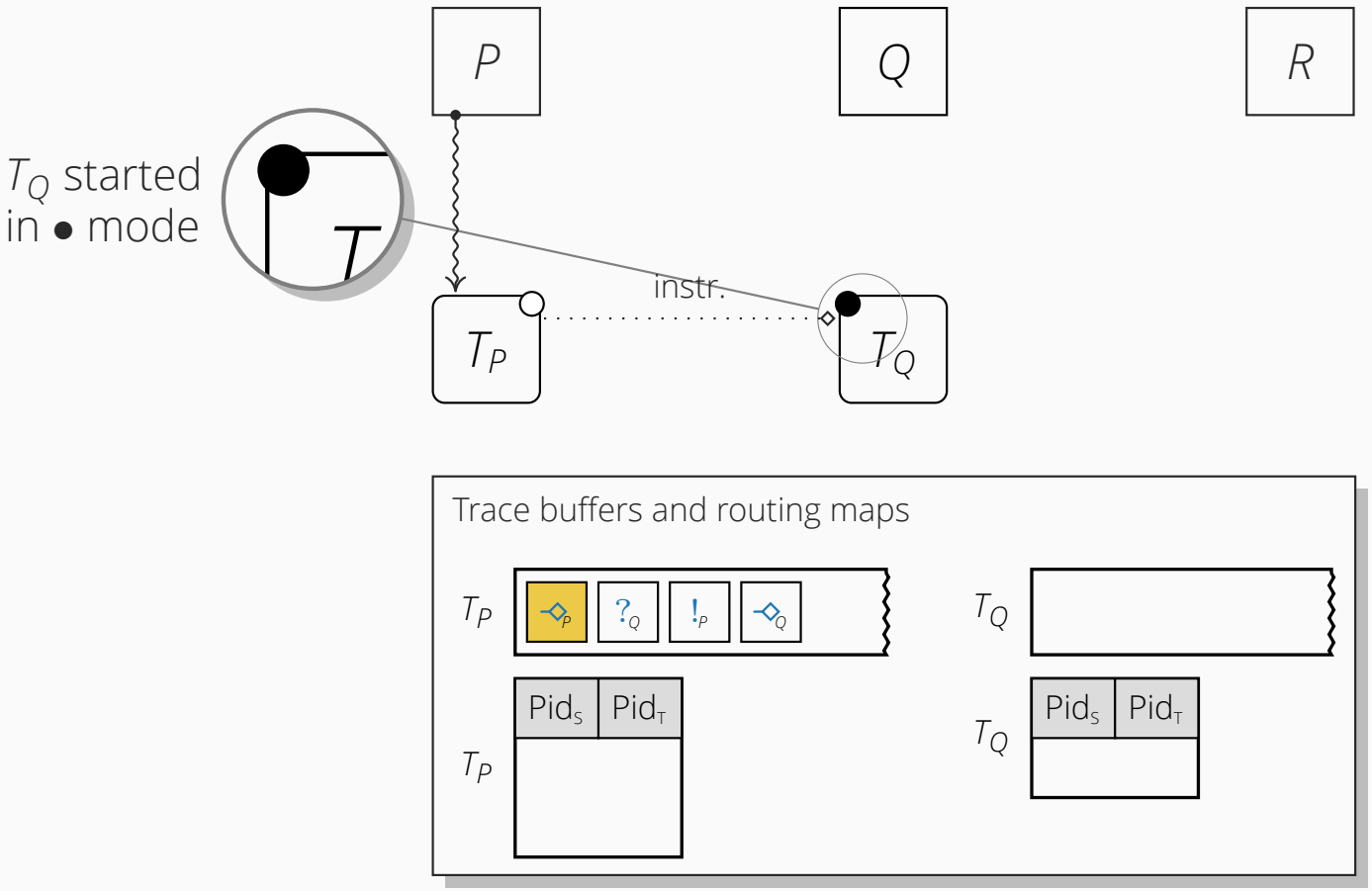
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



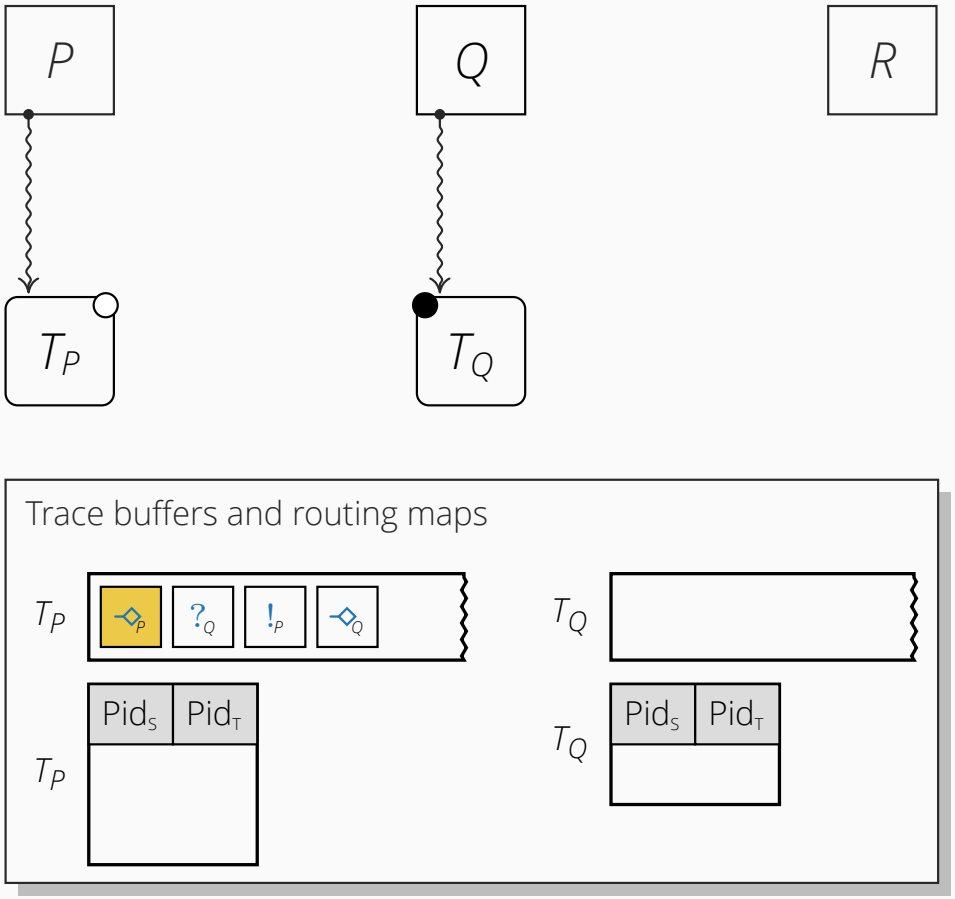
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



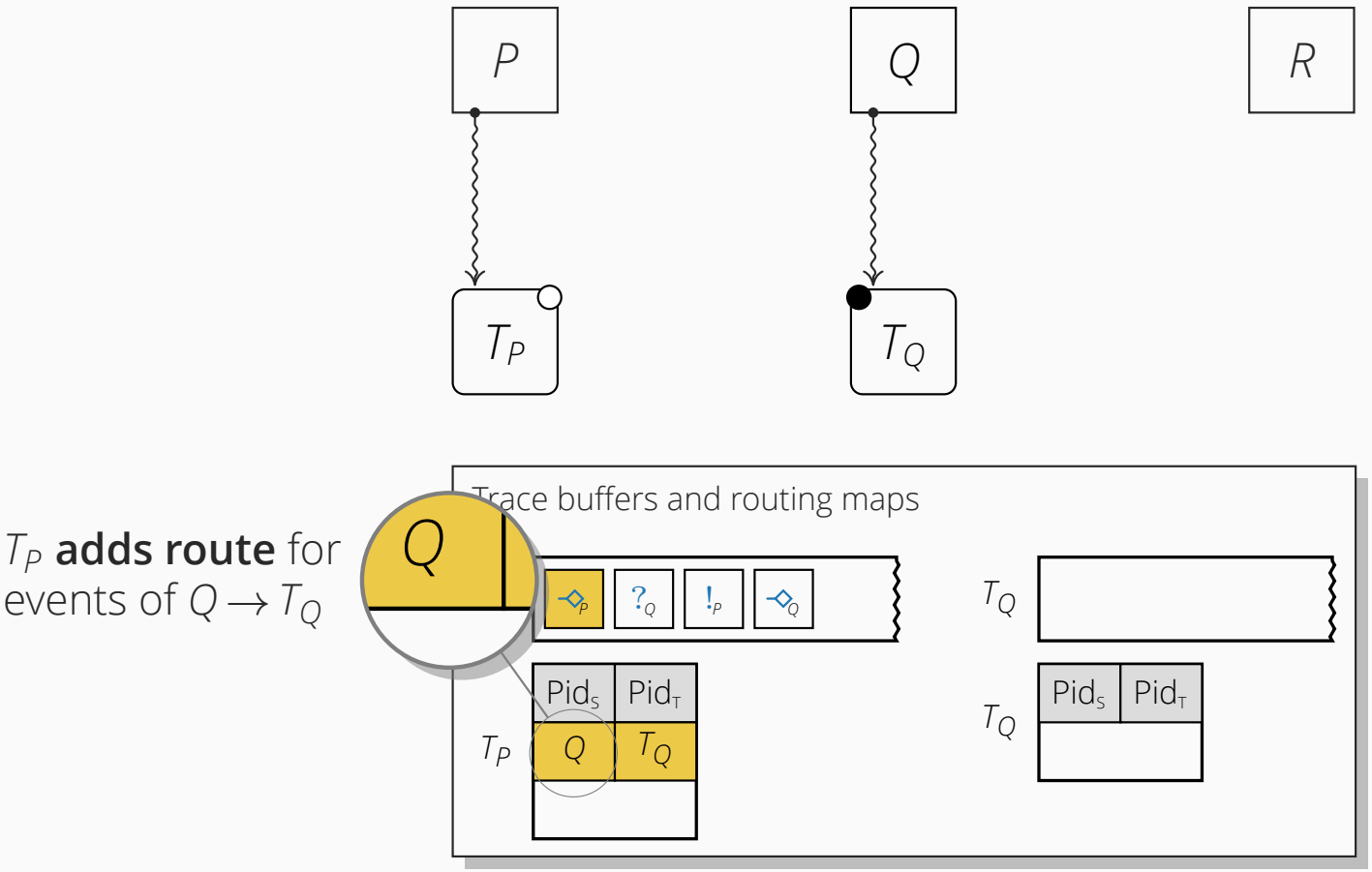
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



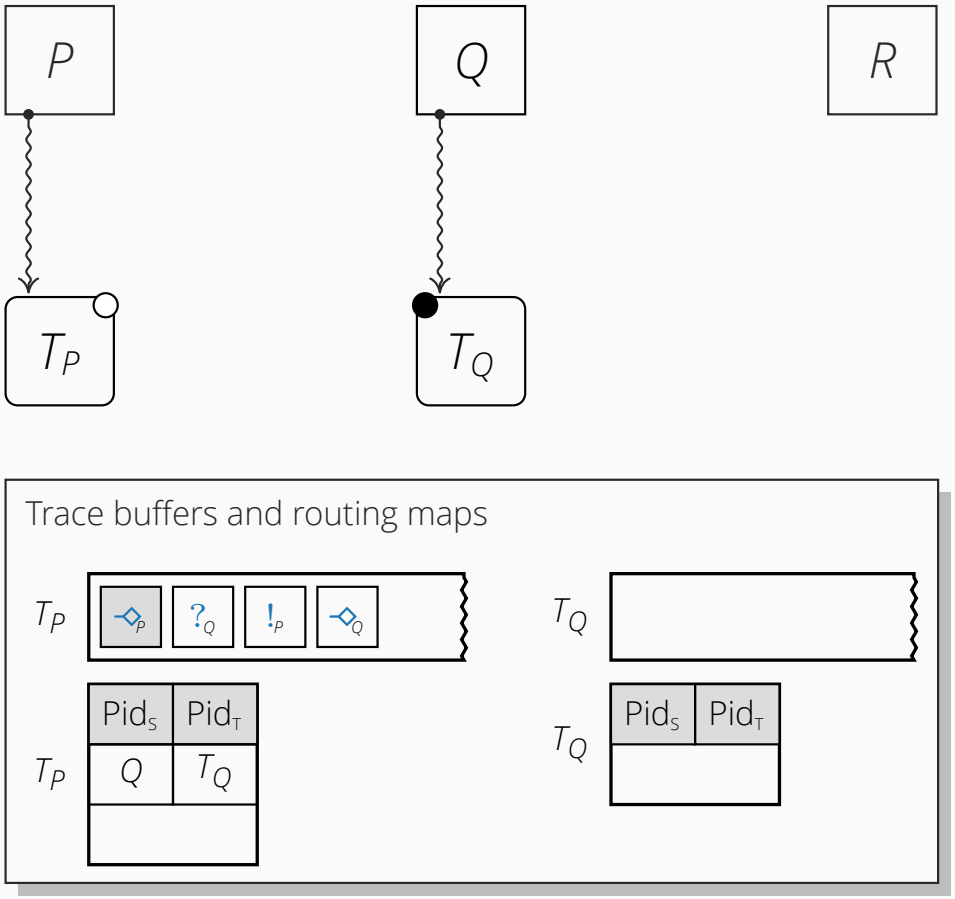
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



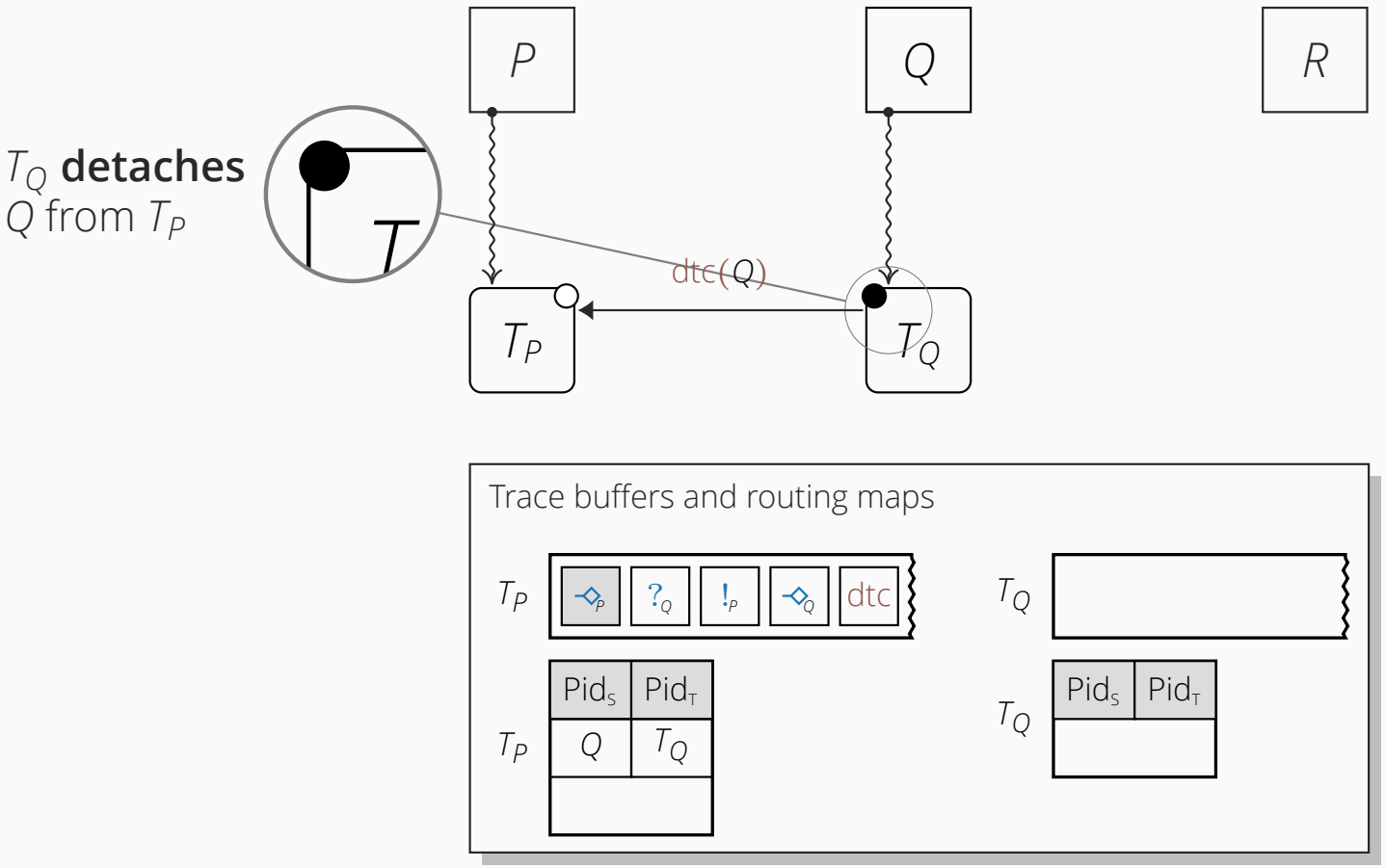
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



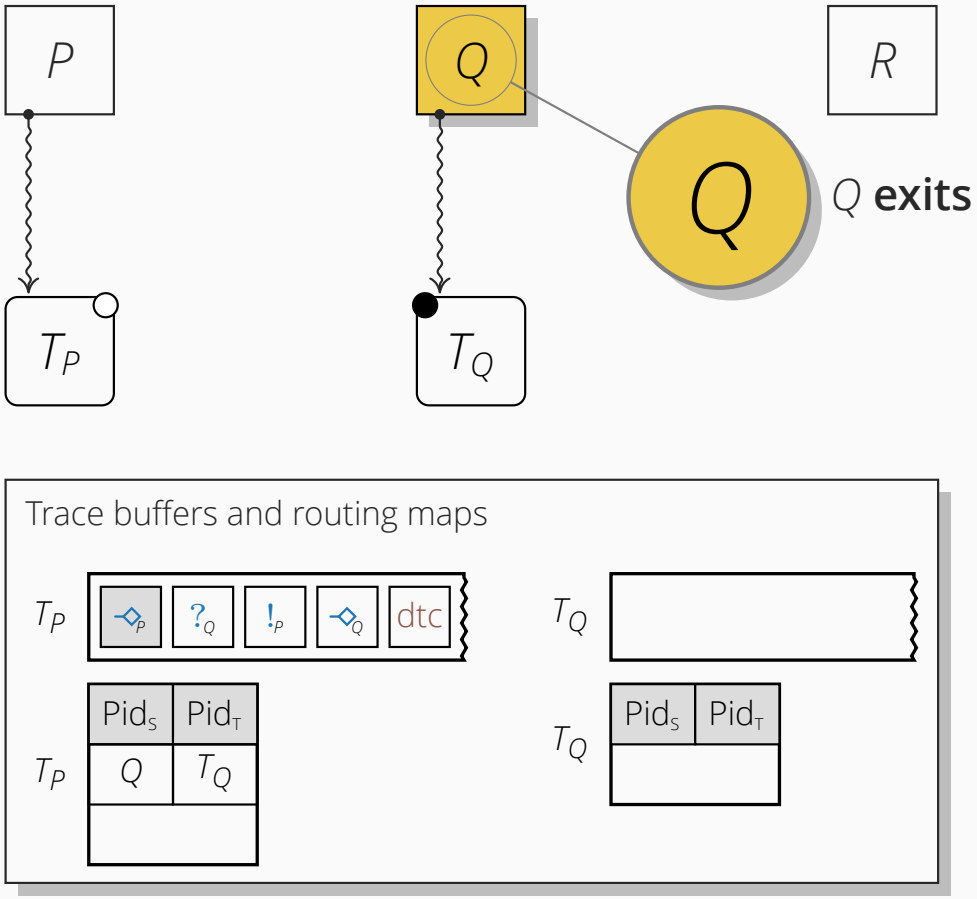
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



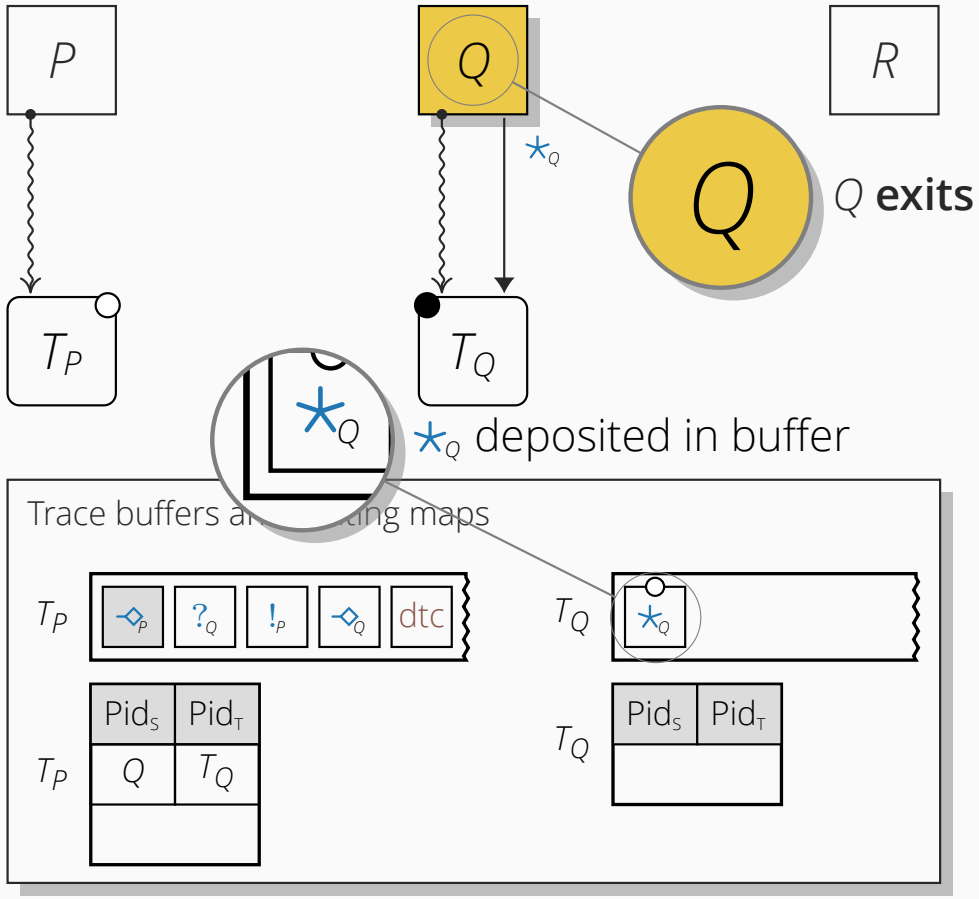
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



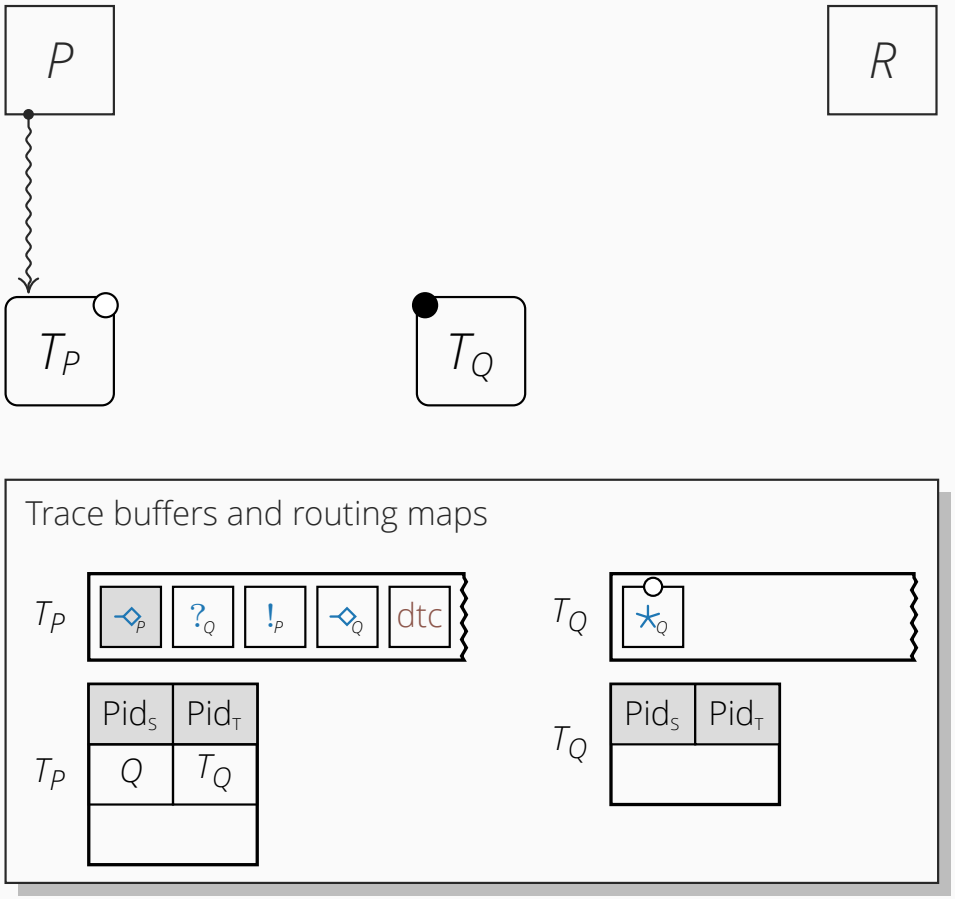
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



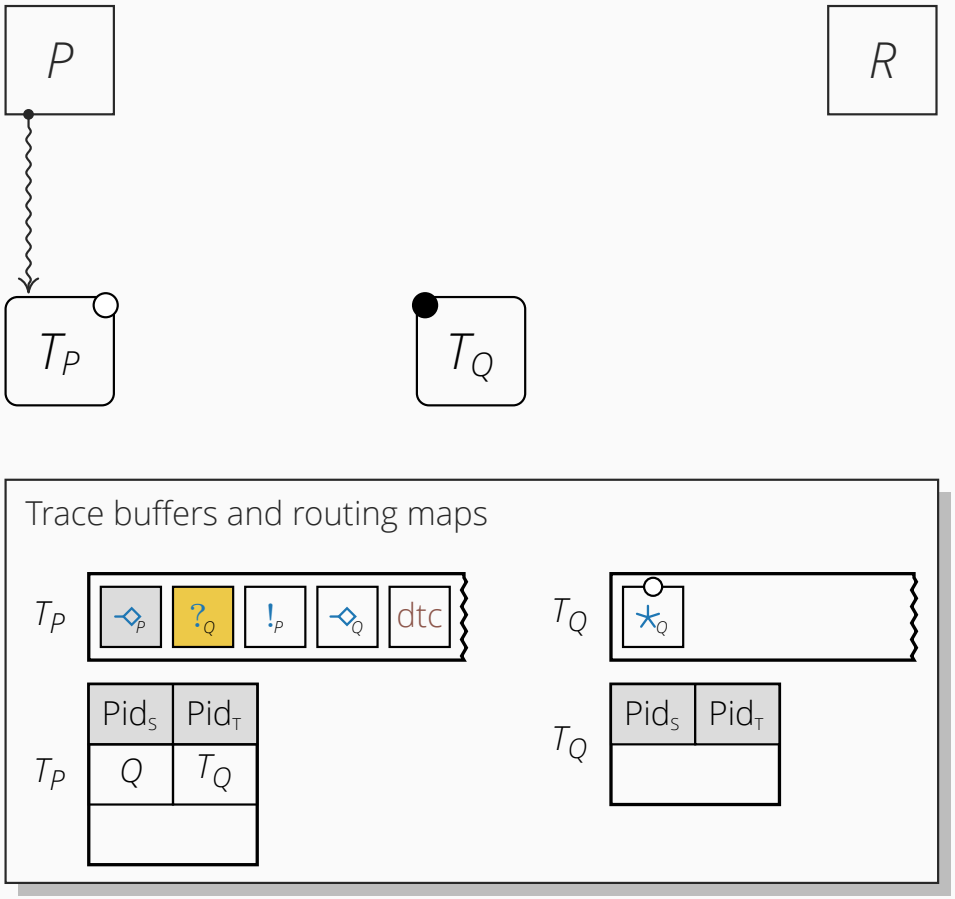
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



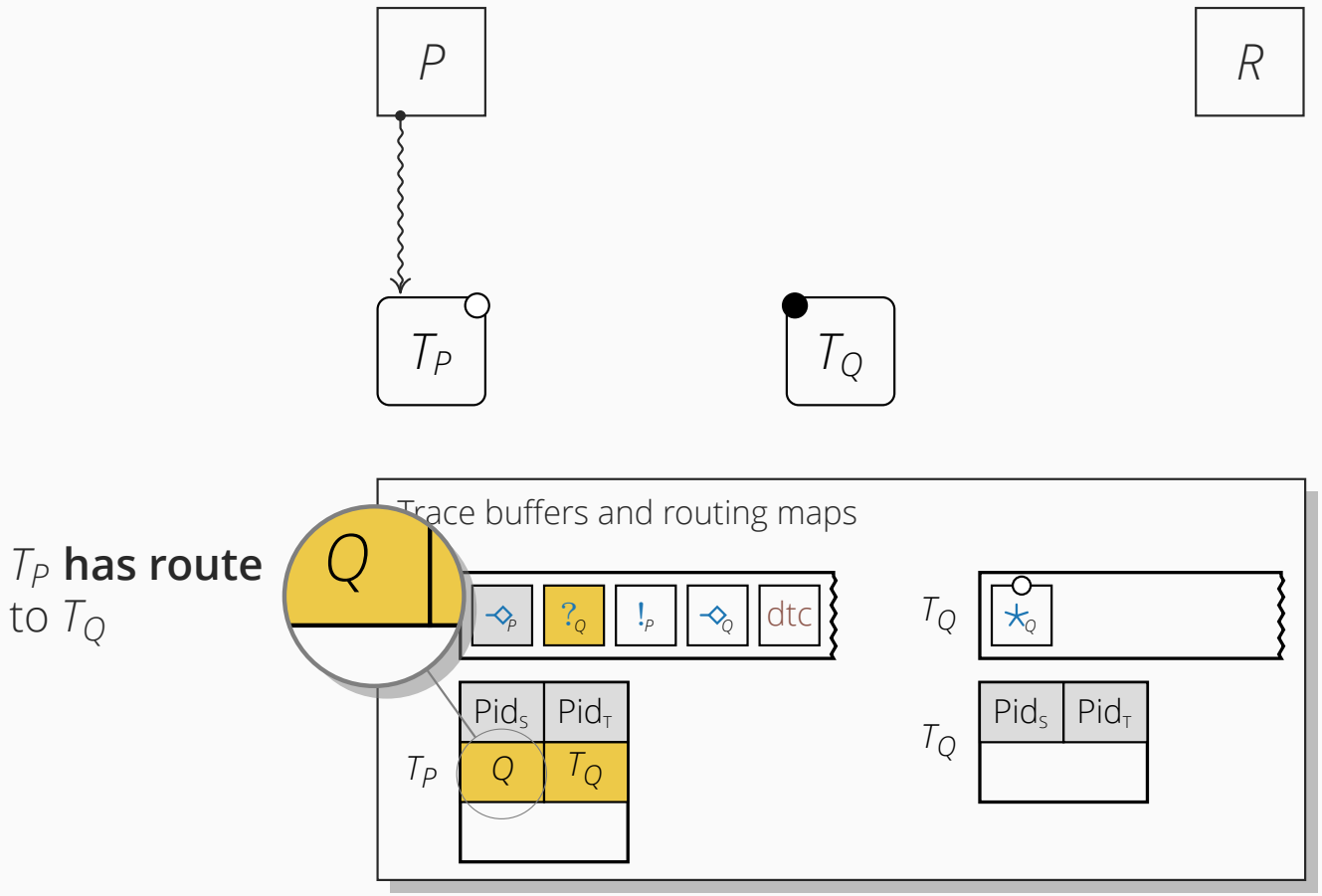
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



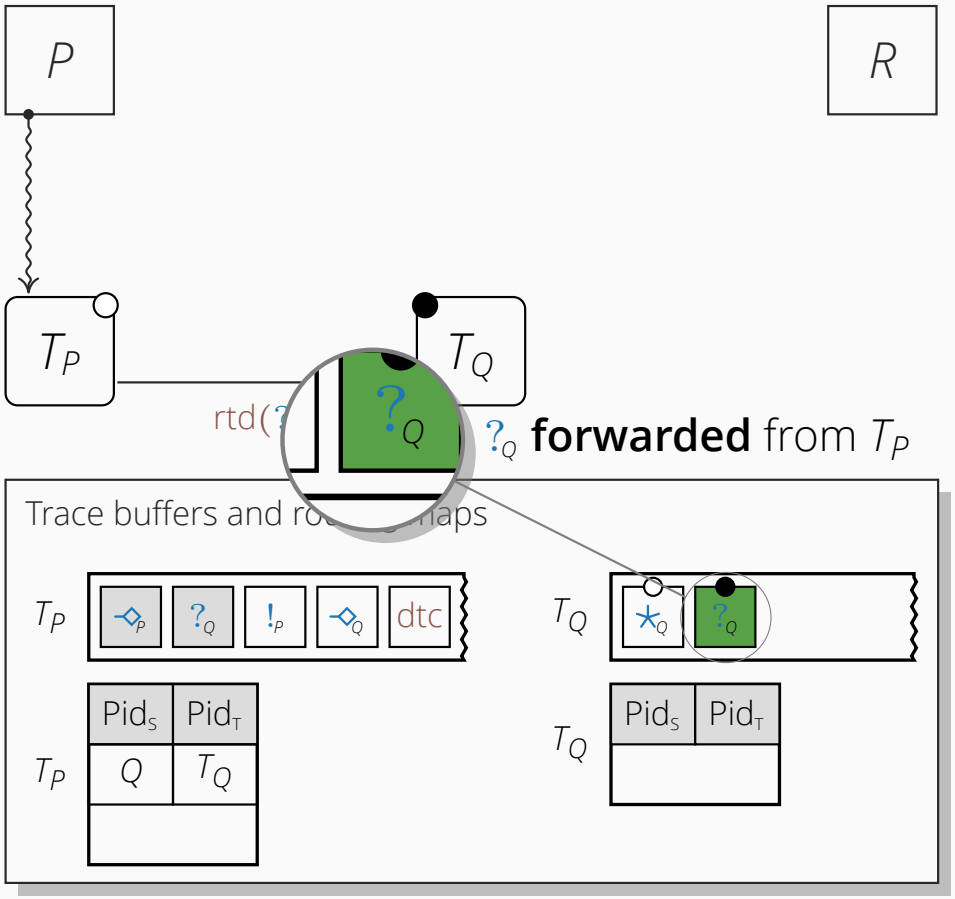
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



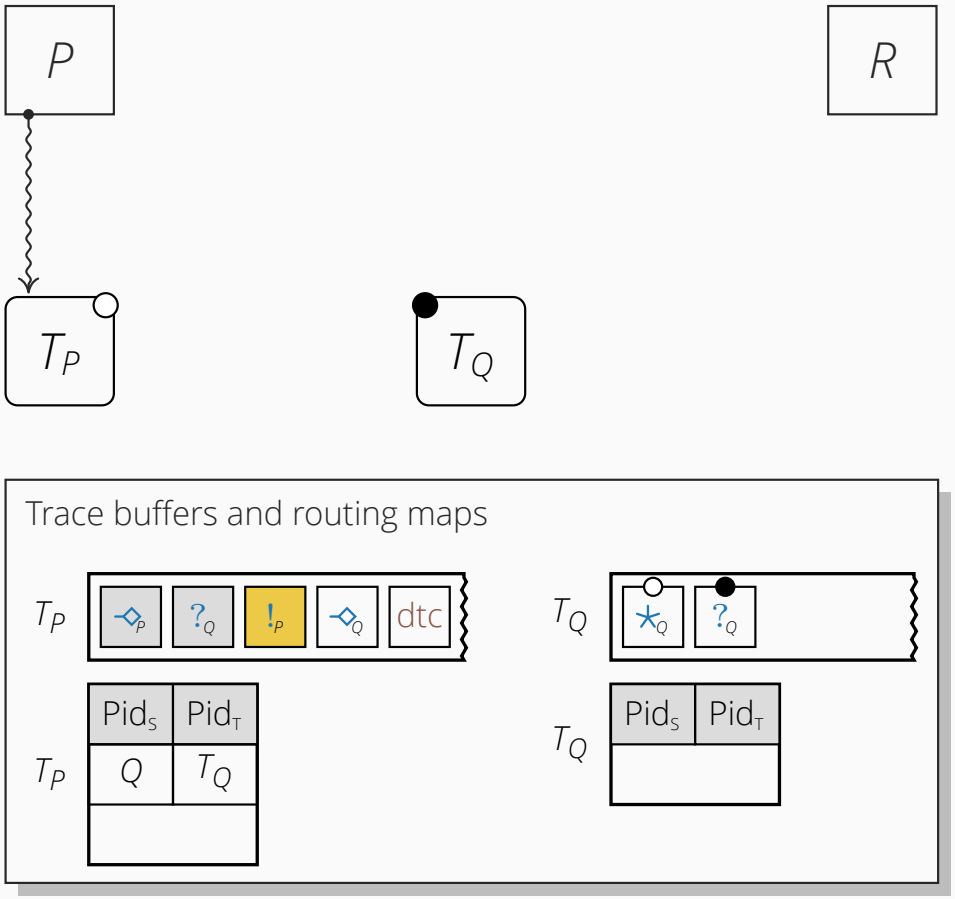
Ensuring consistent traces

Tracers process **forwarded** (•) events before **direct** events (◦)



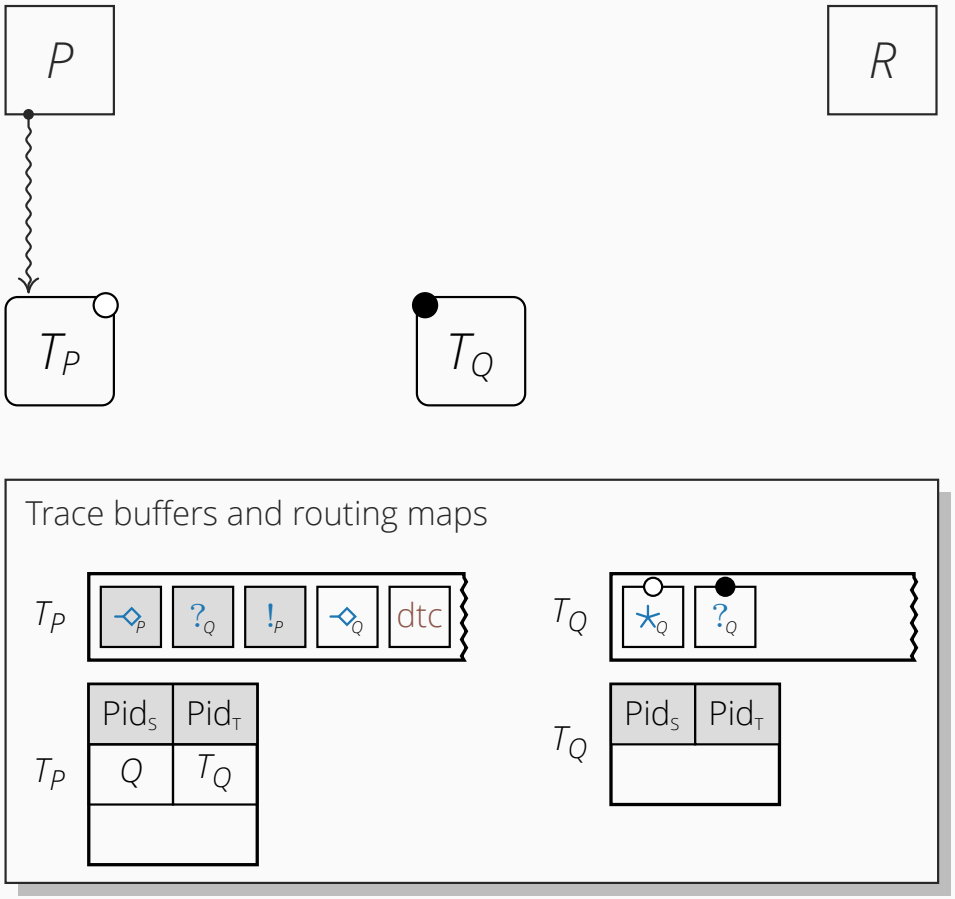
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



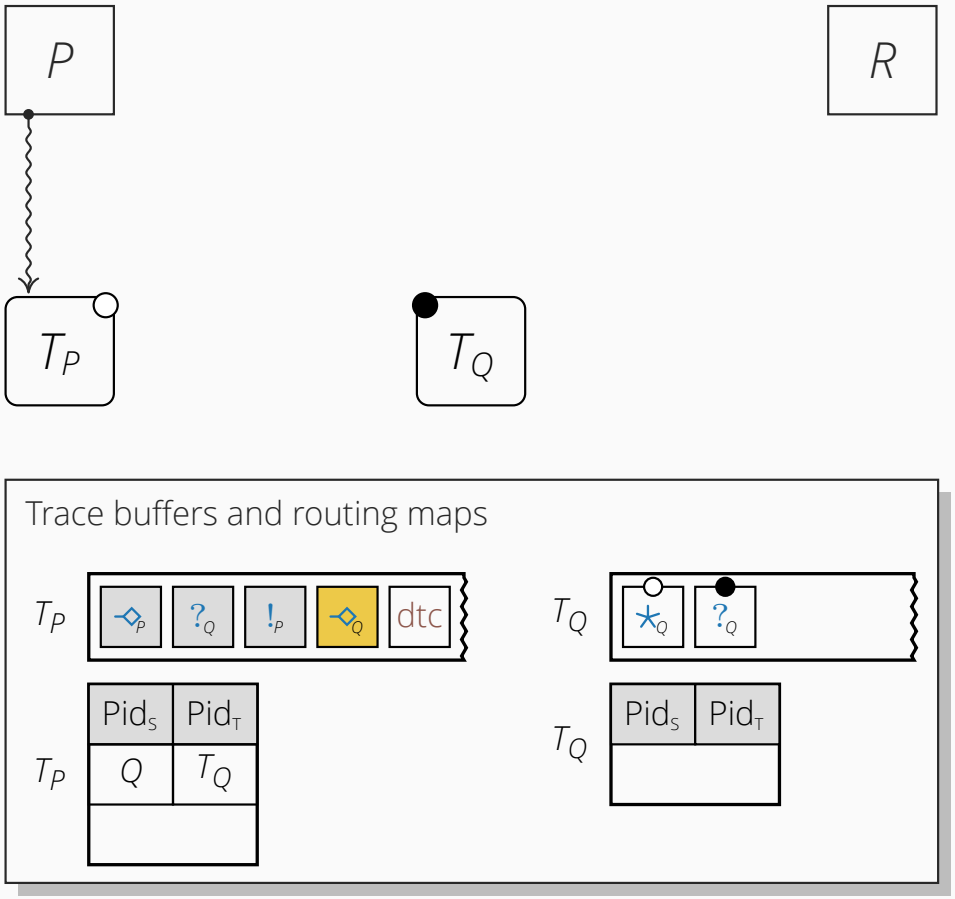
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



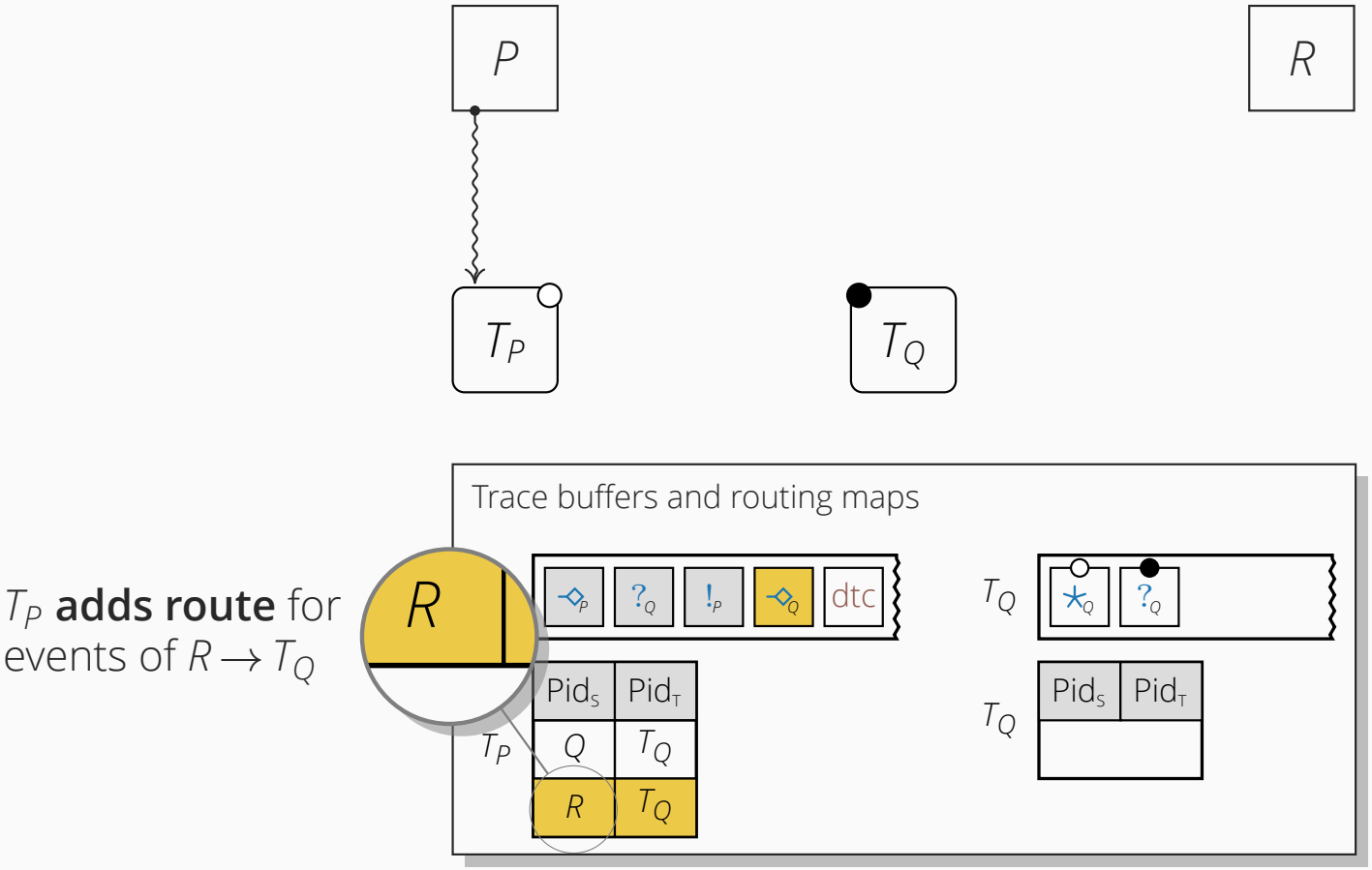
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



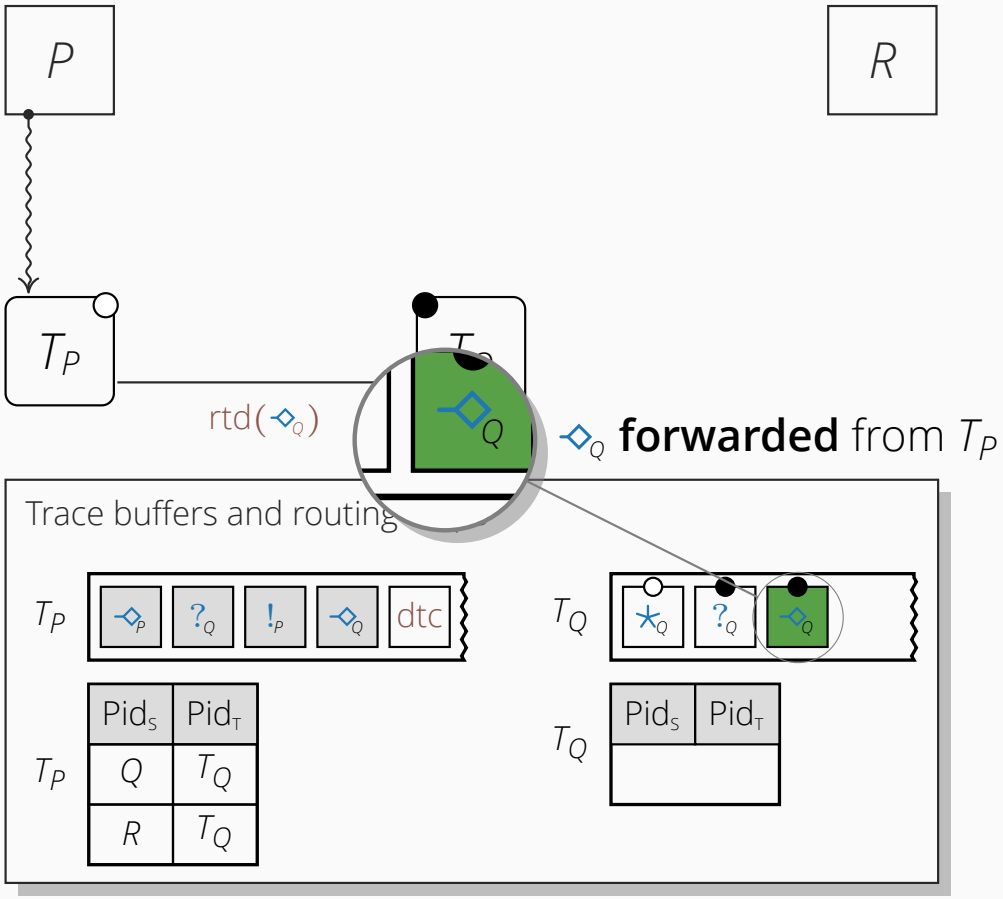
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



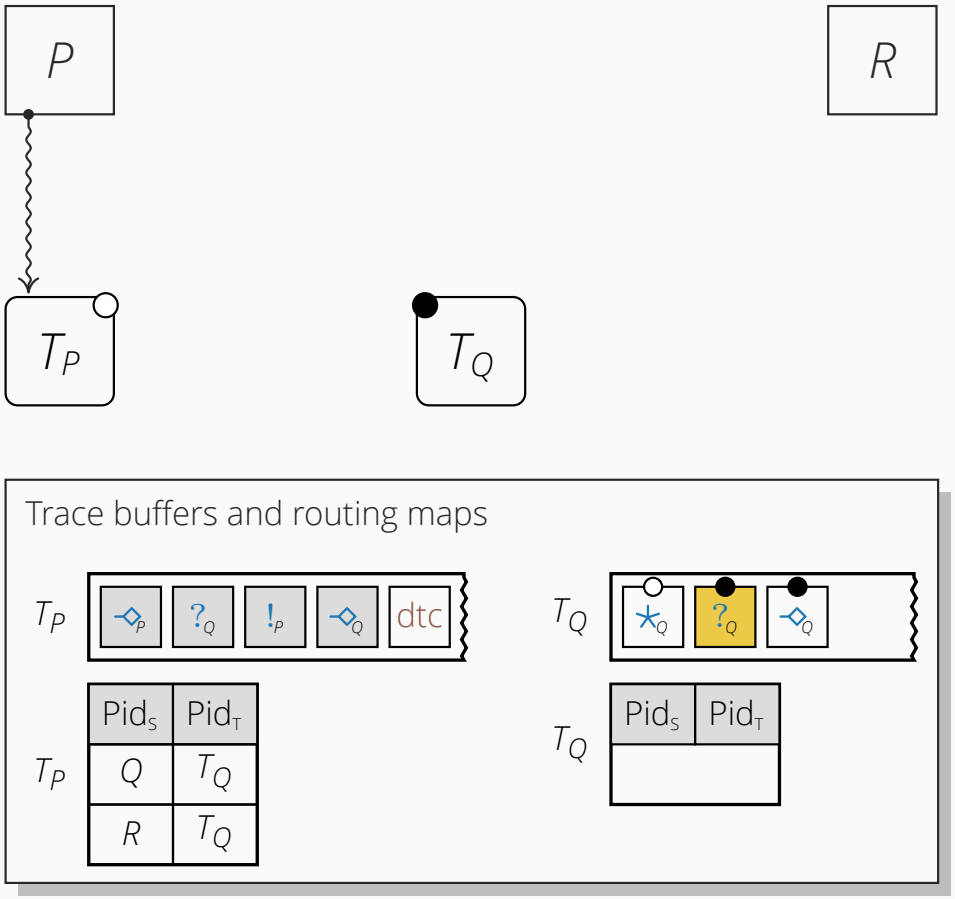
Ensuring consistent traces

Tracers process **forwarded** (•) events before **direct** events (○)



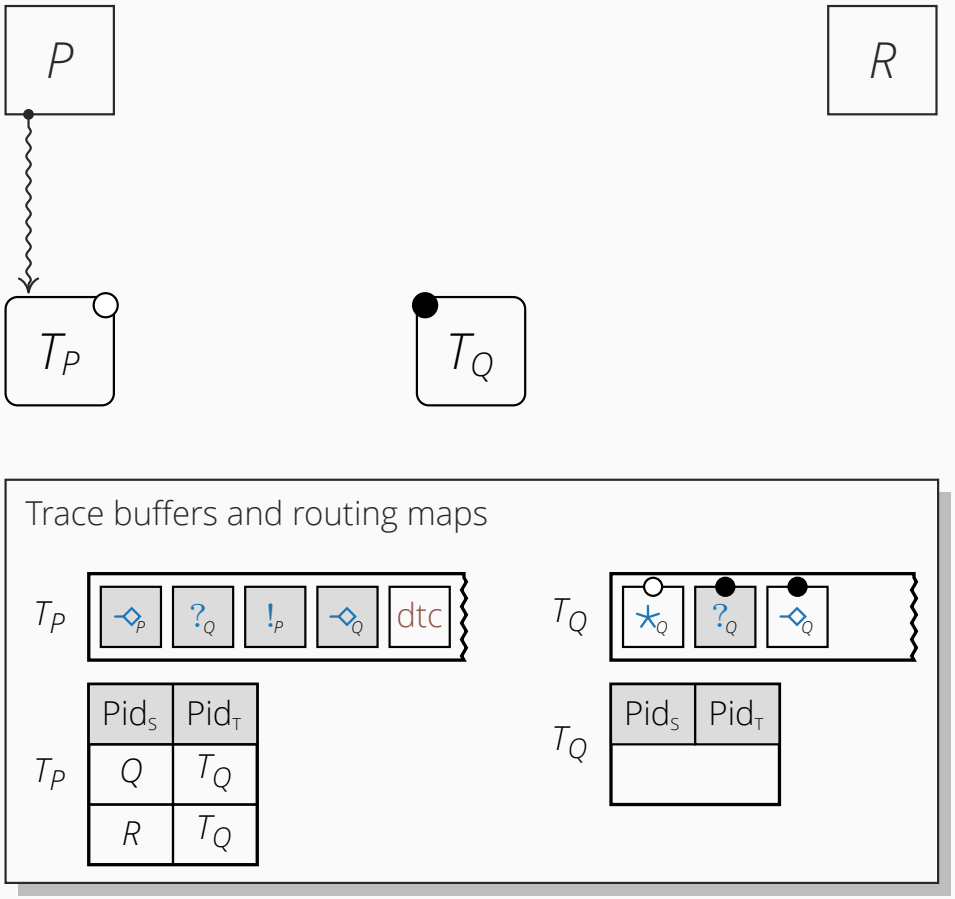
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



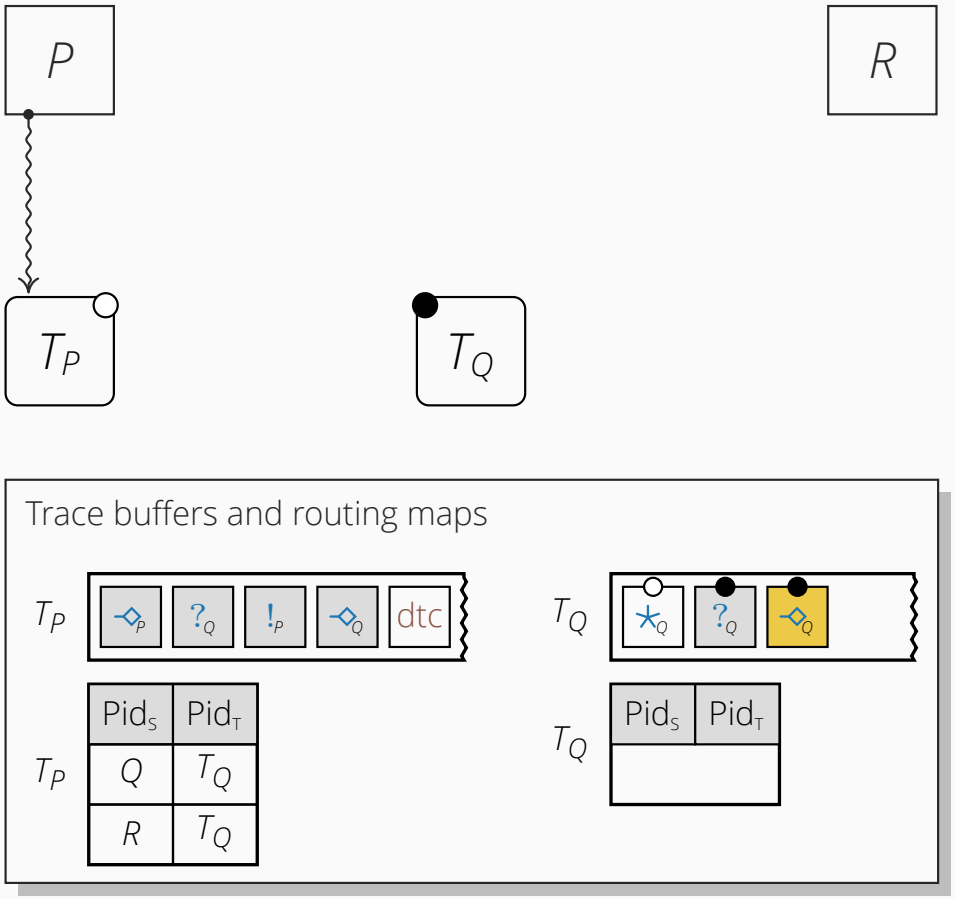
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



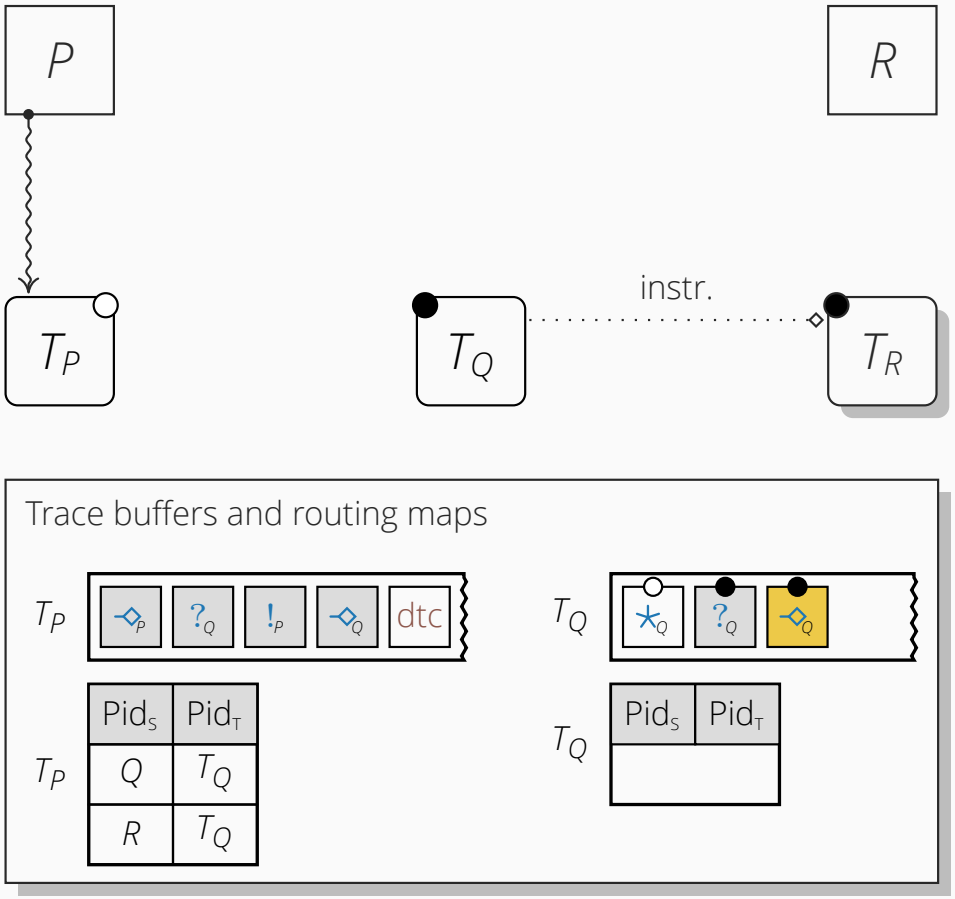
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



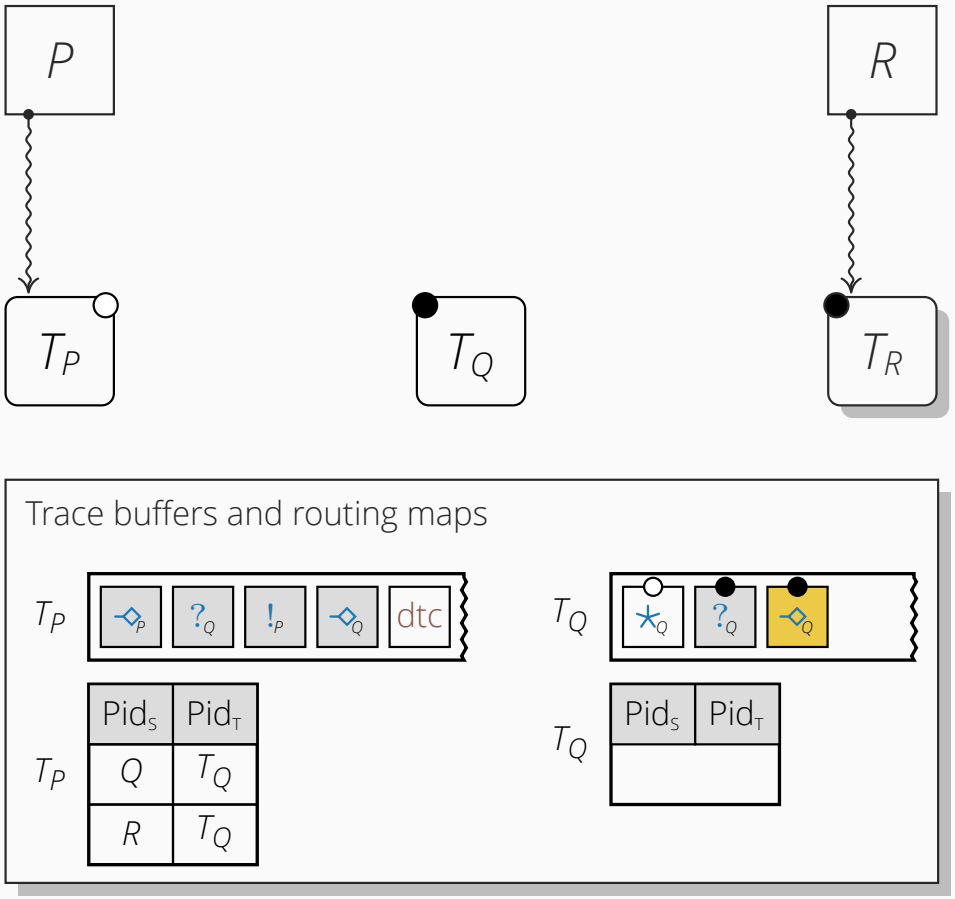
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



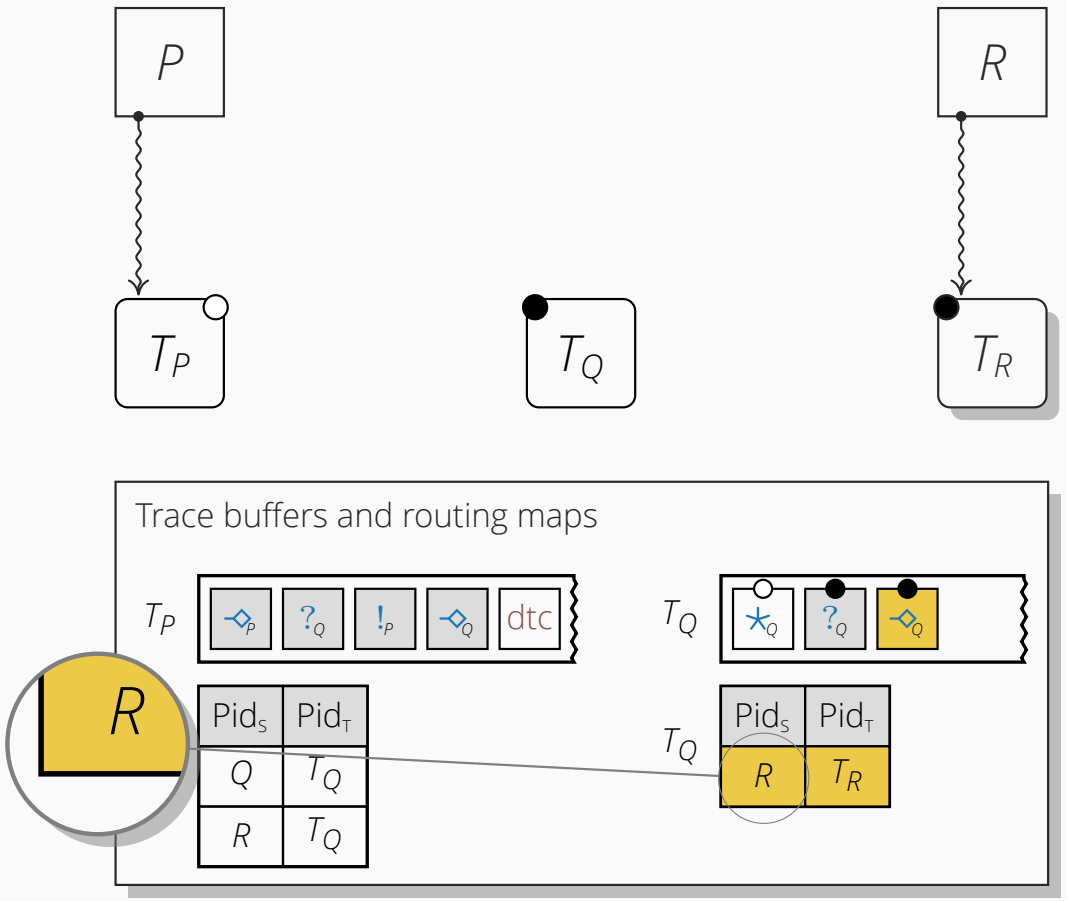
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



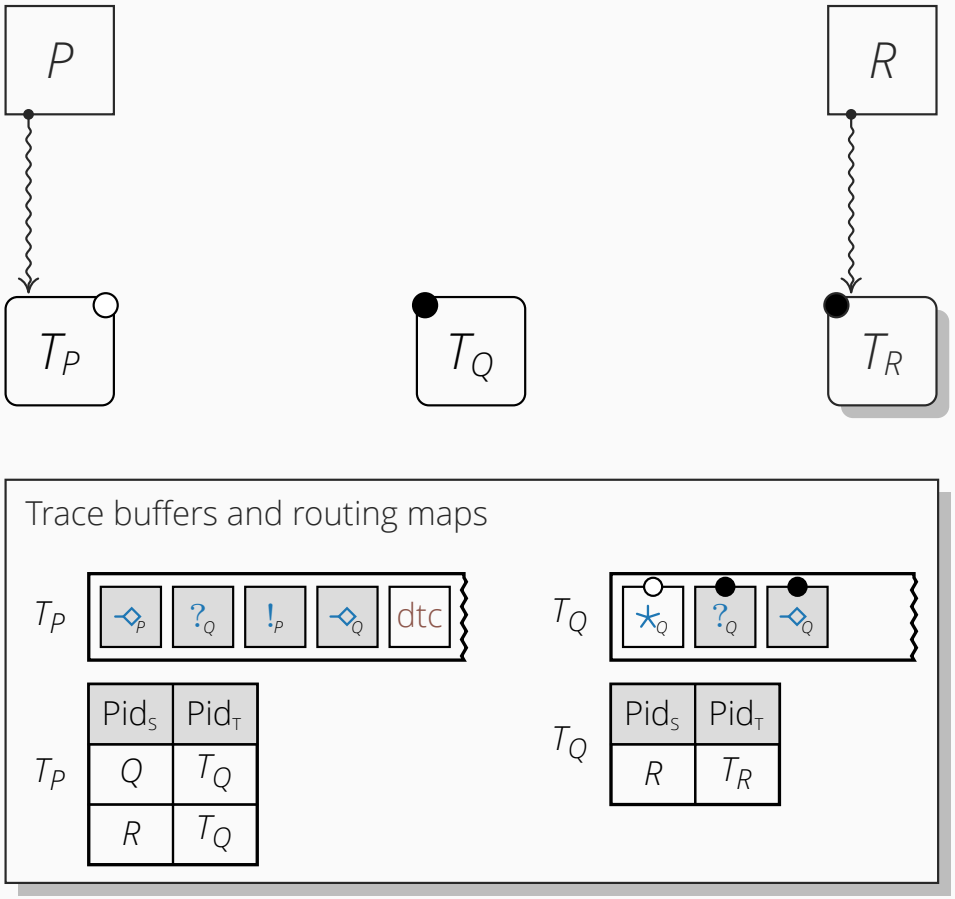
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



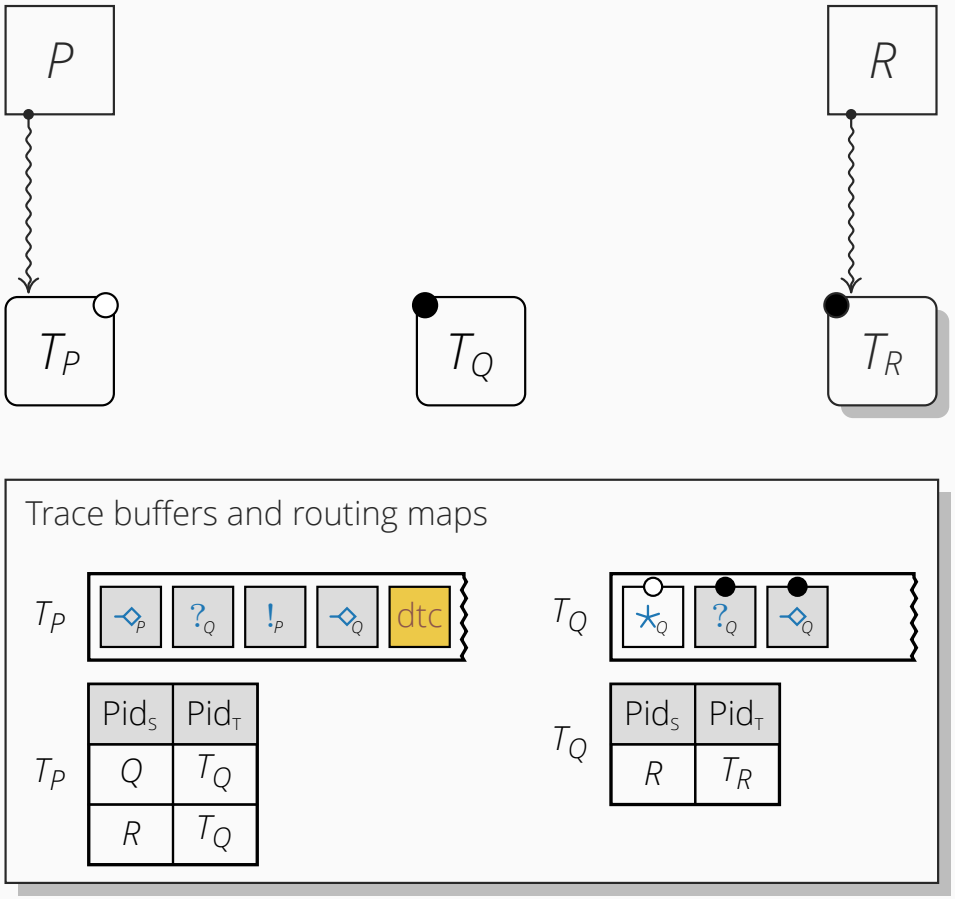
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



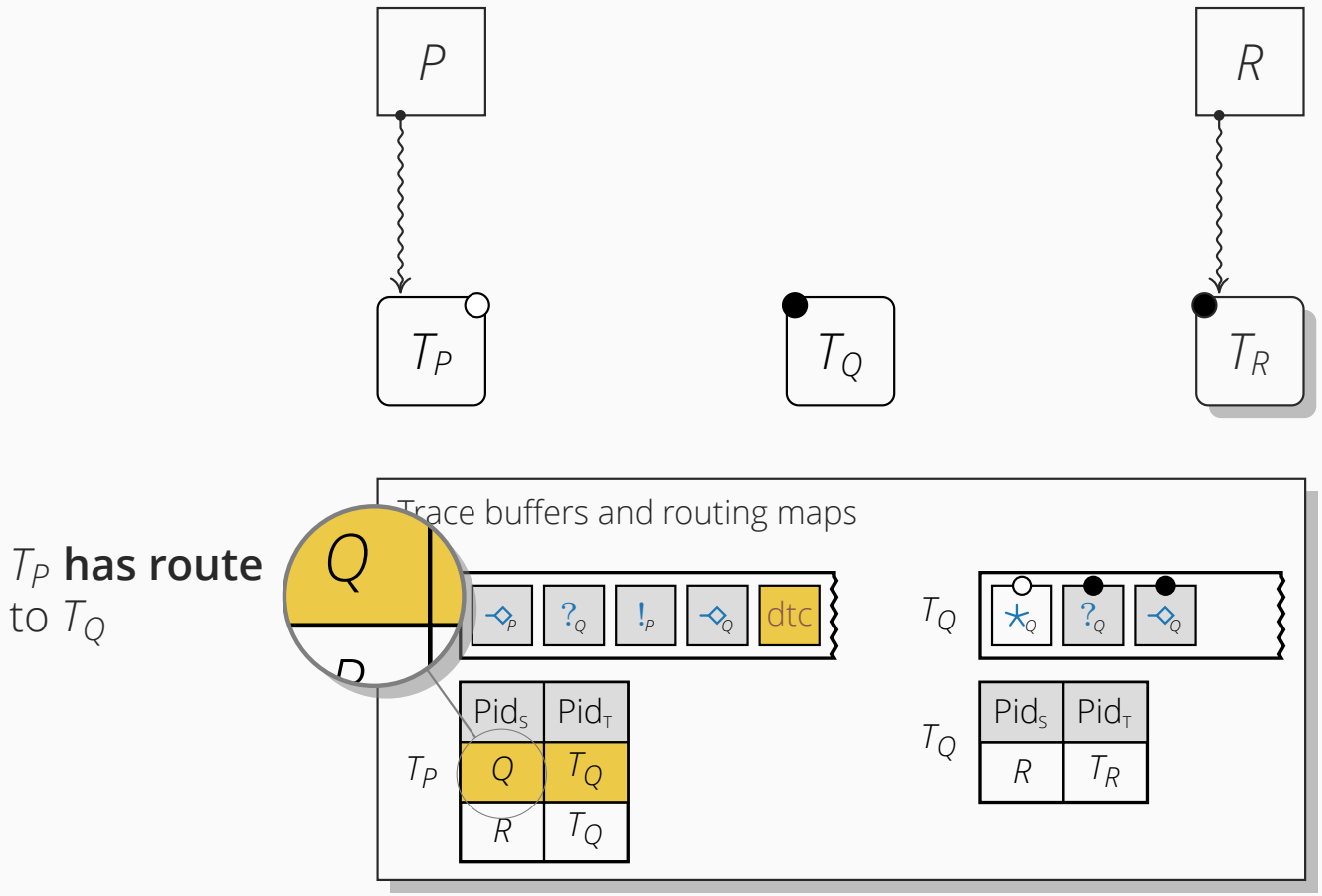
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



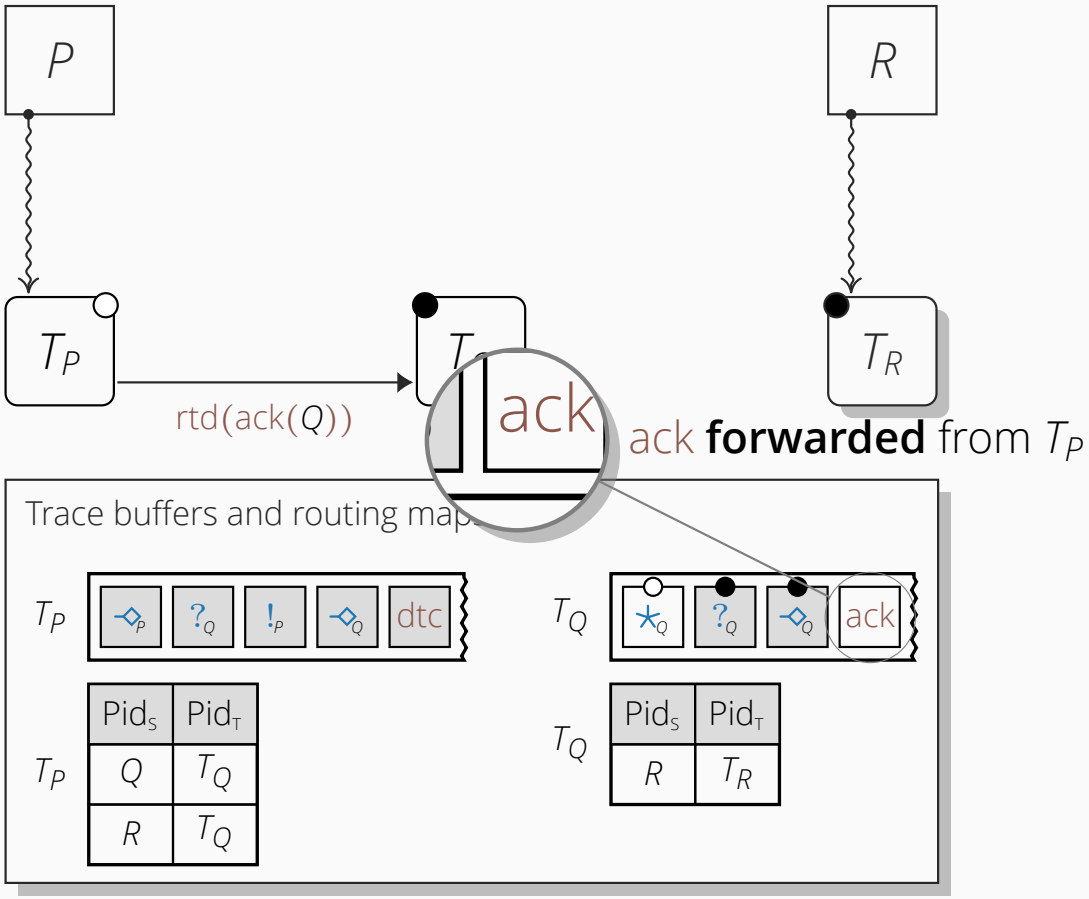
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



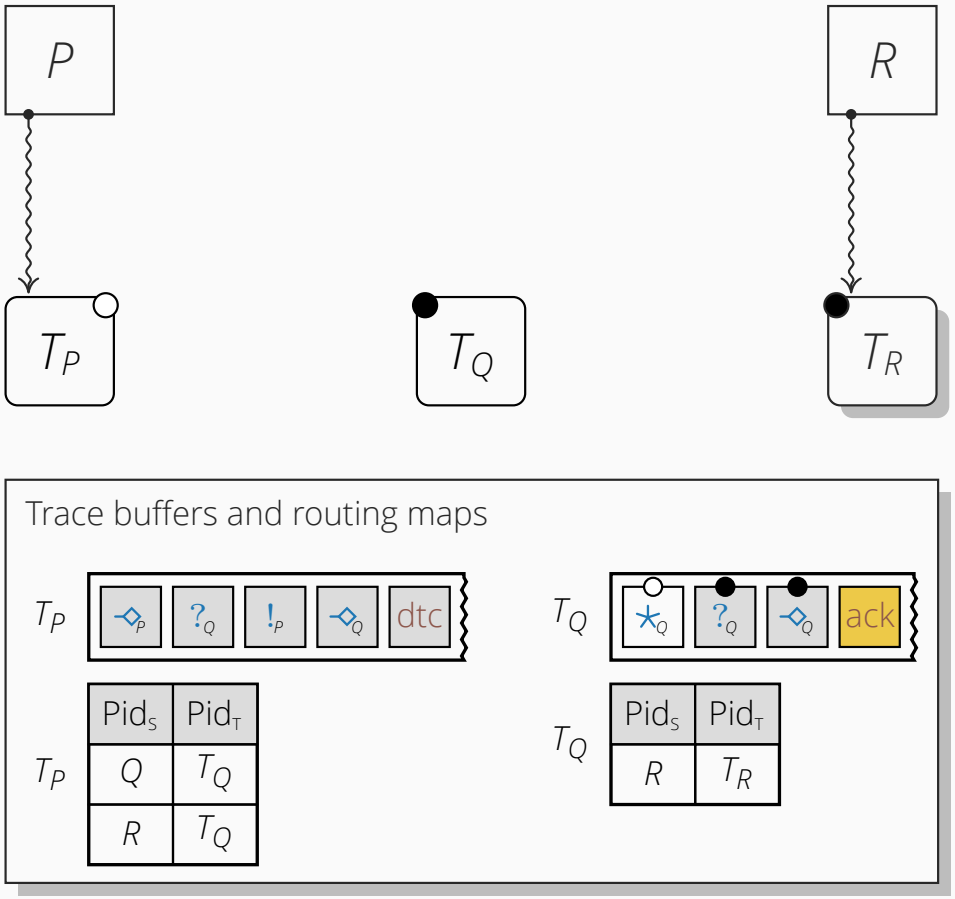
Ensuring consistent traces

Tracers process **forwarded** (•) events before **direct** events (◦)



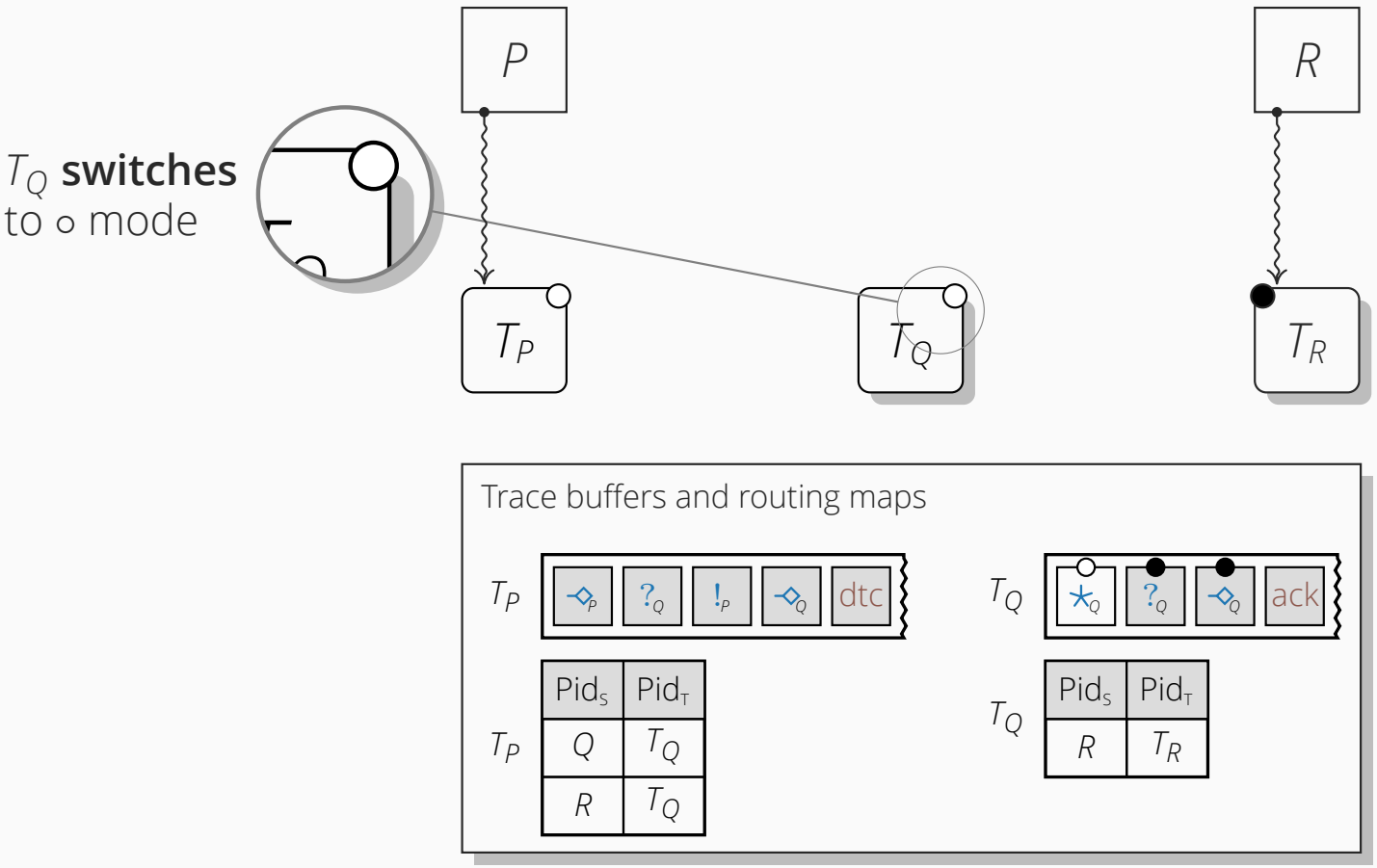
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



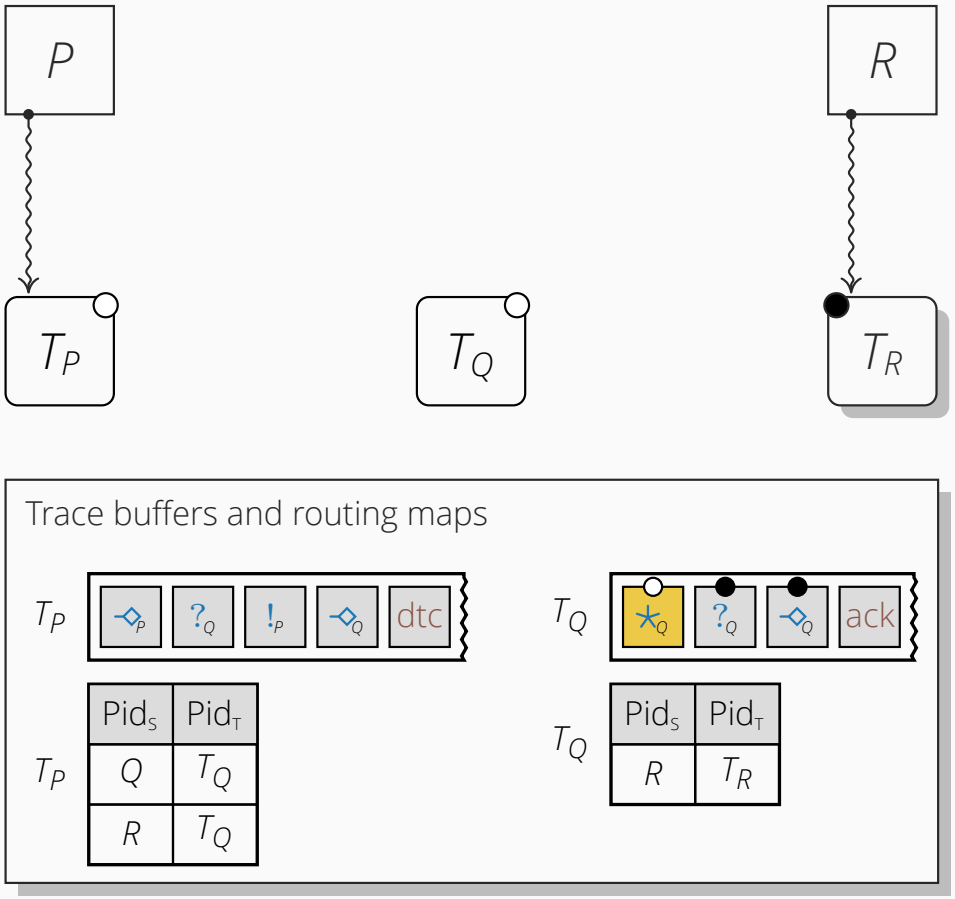
Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



Ensuring consistent traces

Tracers process **forwarded** (●) events before **direct** events (○)



Evlauating RIARC

L. Aceto · **D. P. Attard** · A. Francalanza · A. Ingólfsdóttir · RU · GSSI · UM · U*of*G

Evlauating RIARC

Implementability

To confirm that RIARC can be used in **practice**

Evaluating RIARC

Implementability

To confirm that RIARC can be used in **practice**

Correctness

To confirm that all traces RIARC reports are **sound**

Evlauating RIARC

Implementability

To confirm that RIARC can be used in **practice**

Correctness

To confirm that all traces RIARC reports are **sound**

Performance

To confirm that RIARC preserves the system **reactiveness**

Implementability of RIARC in Erlang

Erlang: specifically built for **soft real-time** reactive systems

RIARC tracers naturally map to Erlang **actors**

Trace buffers coincide with Erlang **mailboxes**

Erlang has a flexible **tracing infrastructure** meeting A1 - A4

RIARC monitoring modes

- **Online:** actively monitors the system while it executes
- **Offline:** replays a set of pre-recorded executions
- Core RIARC logic is the **same** for both modes

Correctness validation

> *Aim: to confirm that all traces RIARC reports are **sound***

(no missing events and events are in correct the order)

Method: systematic exploration of all interleavings

- Fit our RIARC implementation with assertion **invariants**
- Take all **sound permutations** of system executions
- **Replay** these permutations using our offline engine
- Use monitors that assert the **expected event sequence**

Empirical evaluation

> *Aim: to confirm that RIARC preserves the system* ***reactiveness***

(system is Responsive, Resilient, Message-driven, Elastic)

Method: load test realistic system models

- Edge-case platform: capture **limited** hardware
- General-case platform: capture **commodity** hardware
- High concurrency: benchmarks running **short-lived** tasks
- Moderate concurrency: benchmarks running **long-lived** tasks

BenchCRV for Erlang reactive systems

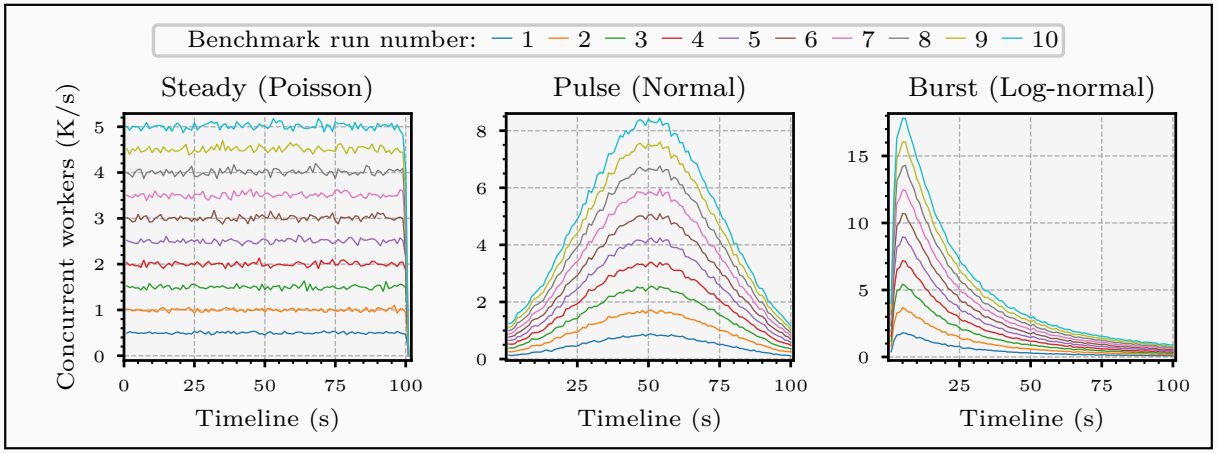
Emulates **real-world** operation of master-worker systems

Latency (ms) | Memory consumption (GB) | Scheduler usage %

Workload shapes

- **Steady:** models system under **stable** workload
- **Pulse:** models system under **rising and falling** workload
- **Burst:** models system under **stress** due to workload spikes

Empirical evaluation set-up



one monitor per process

5μs per event

100s loading time

Platform	Sched.	Concurrency	Workers	Requests	≈ Events
Edge case	4	High	100 k	100	40 M
General case	16	High	500 k	100	200 M
		Moderate	5 k	10 k	

Summary of empirical experiments

Edge-case platform - **40 M** events

High concurrency experiments

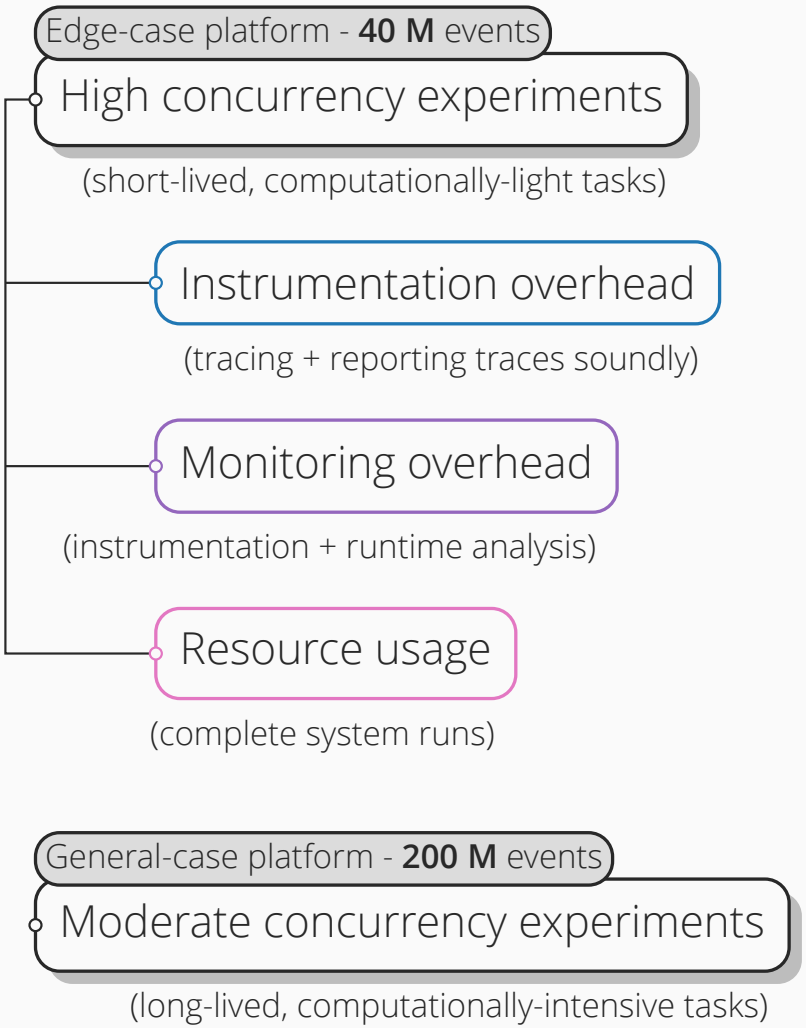
(short-lived, computationally-light tasks)

General-case platform - **200 M** events

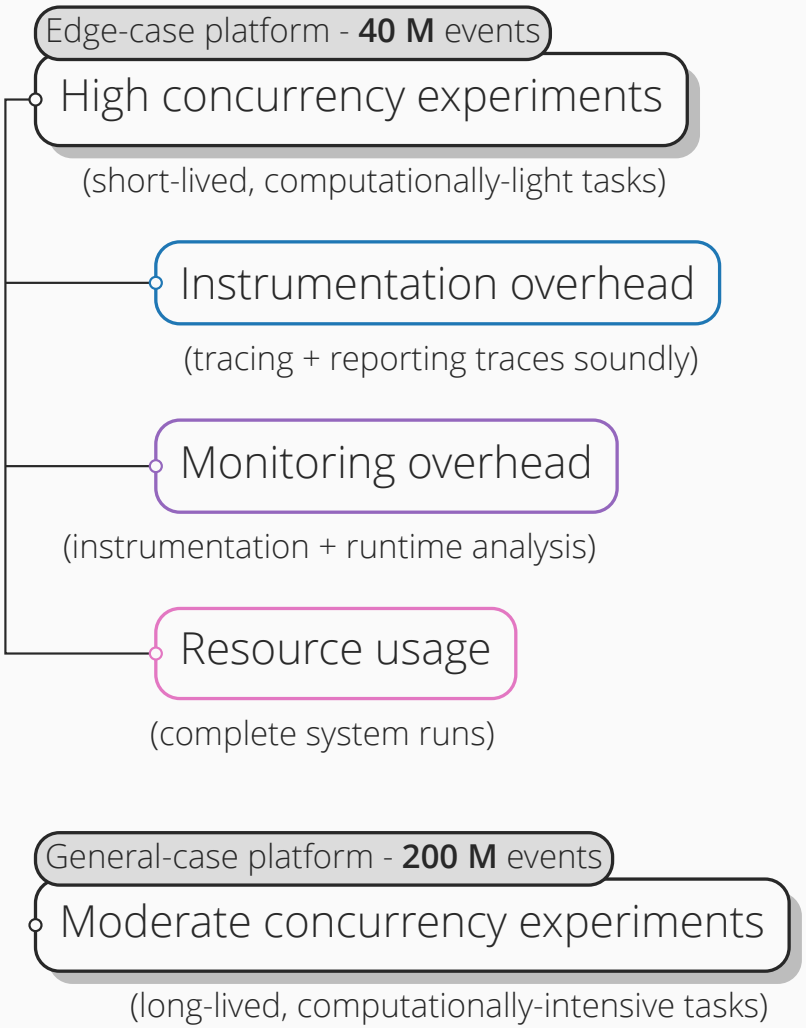
Moderate concurrency experiments

(long-lived, computationally-intensive tasks)

Summary of empirical experiments



Summary of empirical experiments



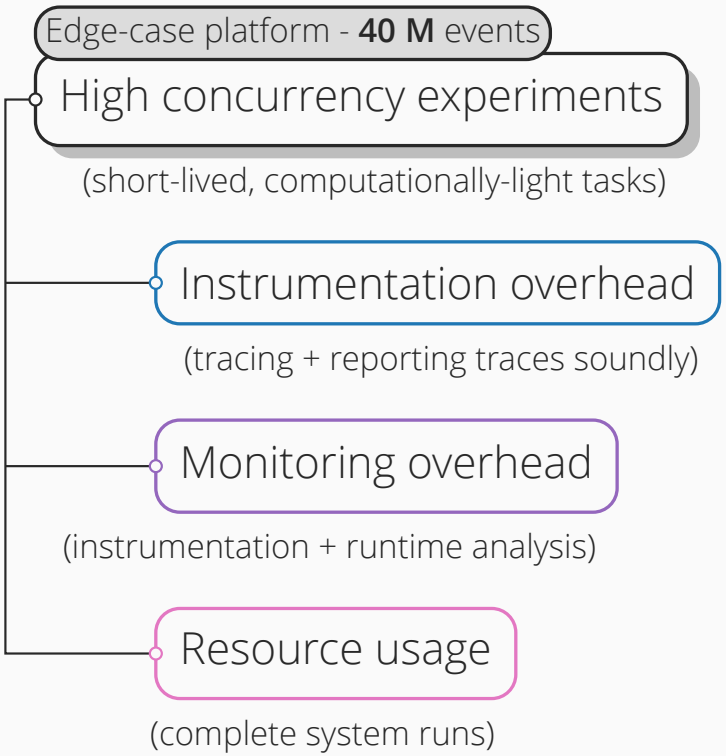
Centralised is **not practical**

Inline is the **most efficient**

RIARC is still **feasible**

Latency of Inline *vs.* RIARC is **small**

Summary of empirical experiments

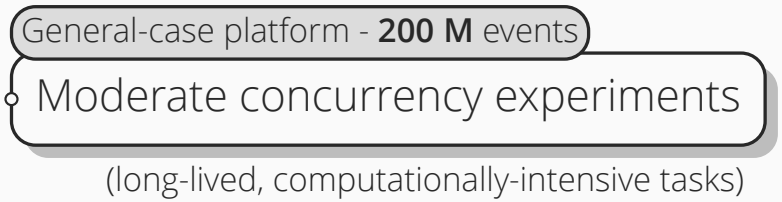


Centralised is **not practical**

Inline is the **most efficient**

RIARC is still **feasible**

Latency of Inline *vs.* RIARC is **small**

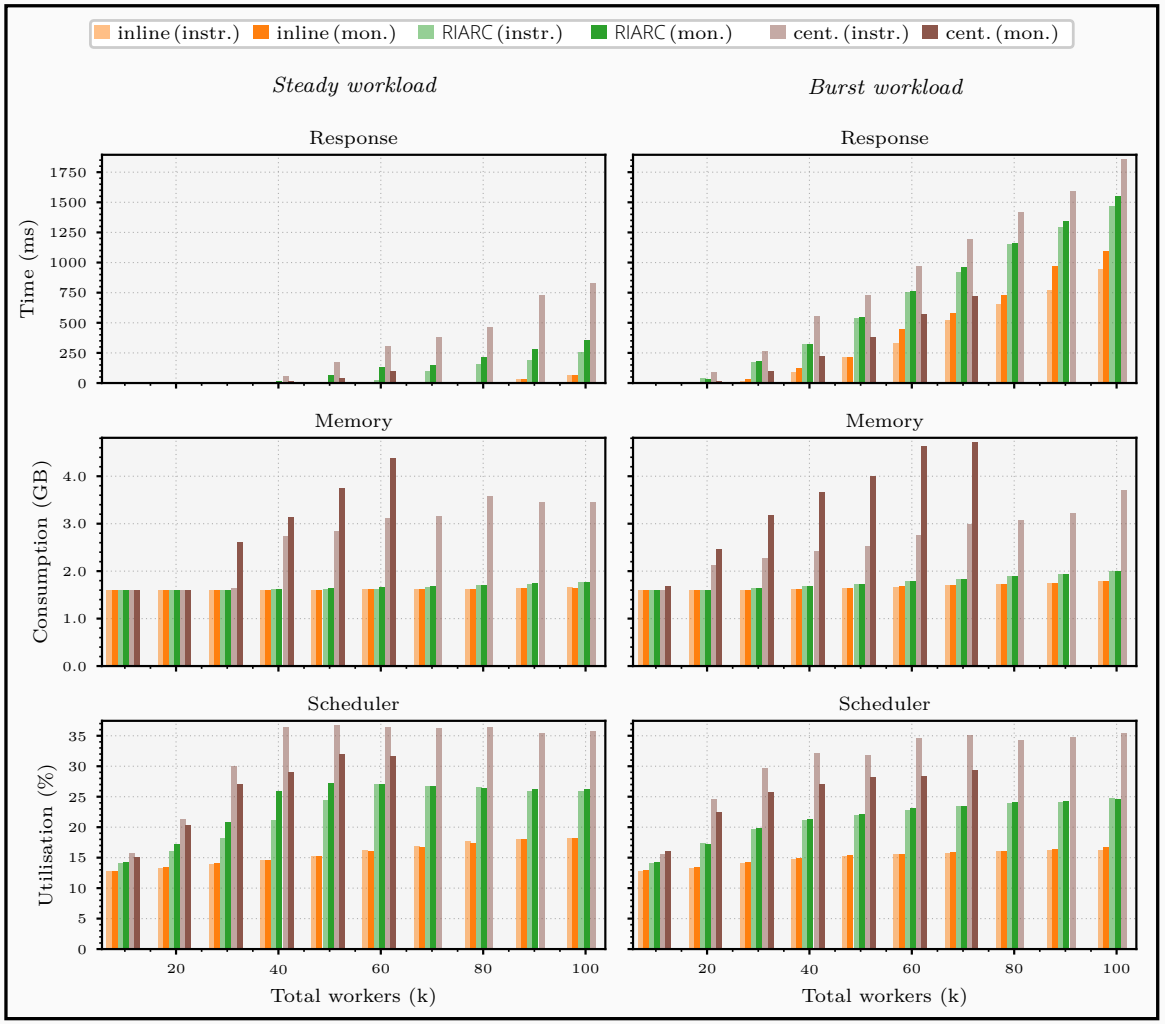


RIARC **scales** to use new schedulers

Inline and RIARC latency **on par**

Inline prone to **slow analysis**

Aim: study overhead effects on the system



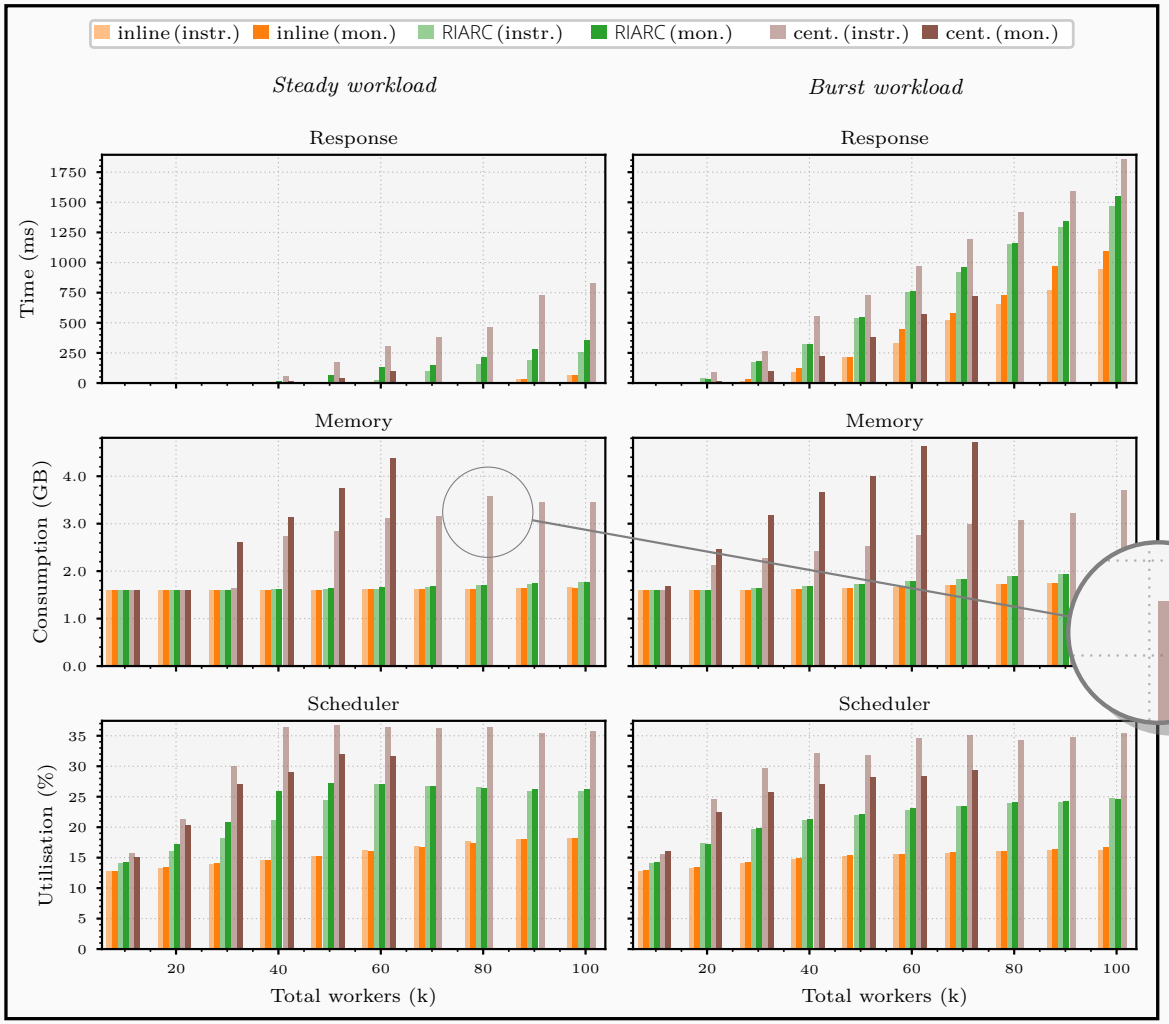
Instrumentation

(tracing + reporting traces soundly)

Monitoring

(instrumentation + runtime analysis)

Aim: study overhead effects on the system



Instrumentation

(tracing + reporting traces soundly)

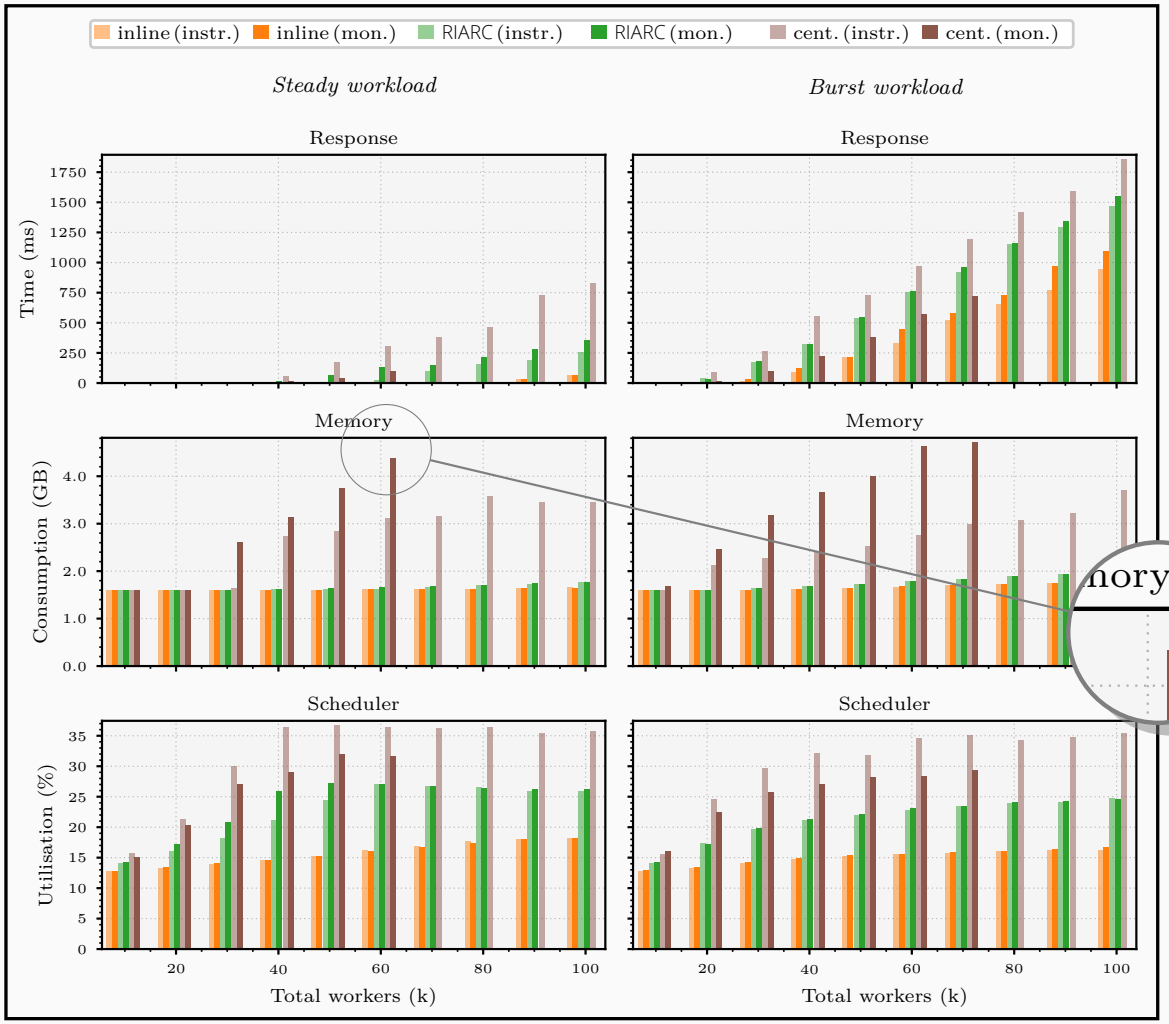
Monitoring

(instrumentation + runtime analysis)

Centralised

Instrumentation yields ↑ memory overhead

Aim: study overhead effects on the system



Instrumentation

(tracing + reporting traces soundly)

Monitoring

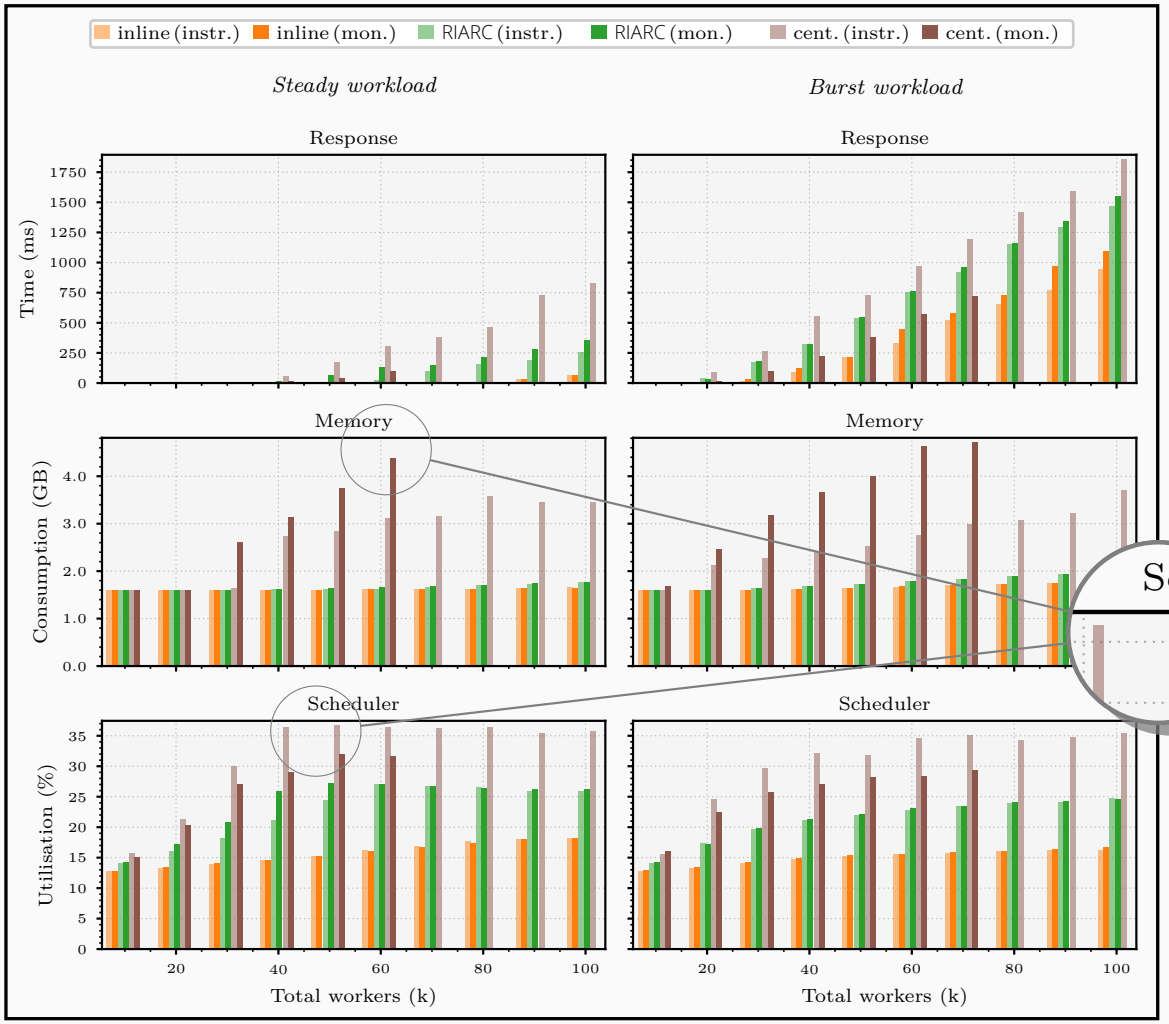
(instrumentation + runtime analysis)

Centralised

Monitoring yields

↑ memory overhead

Aim: study overhead effects on the system



Instrumentation

(tracing + reporting traces soundly)

Monitoring

(instrumentation + runtime analysis)

Centralised

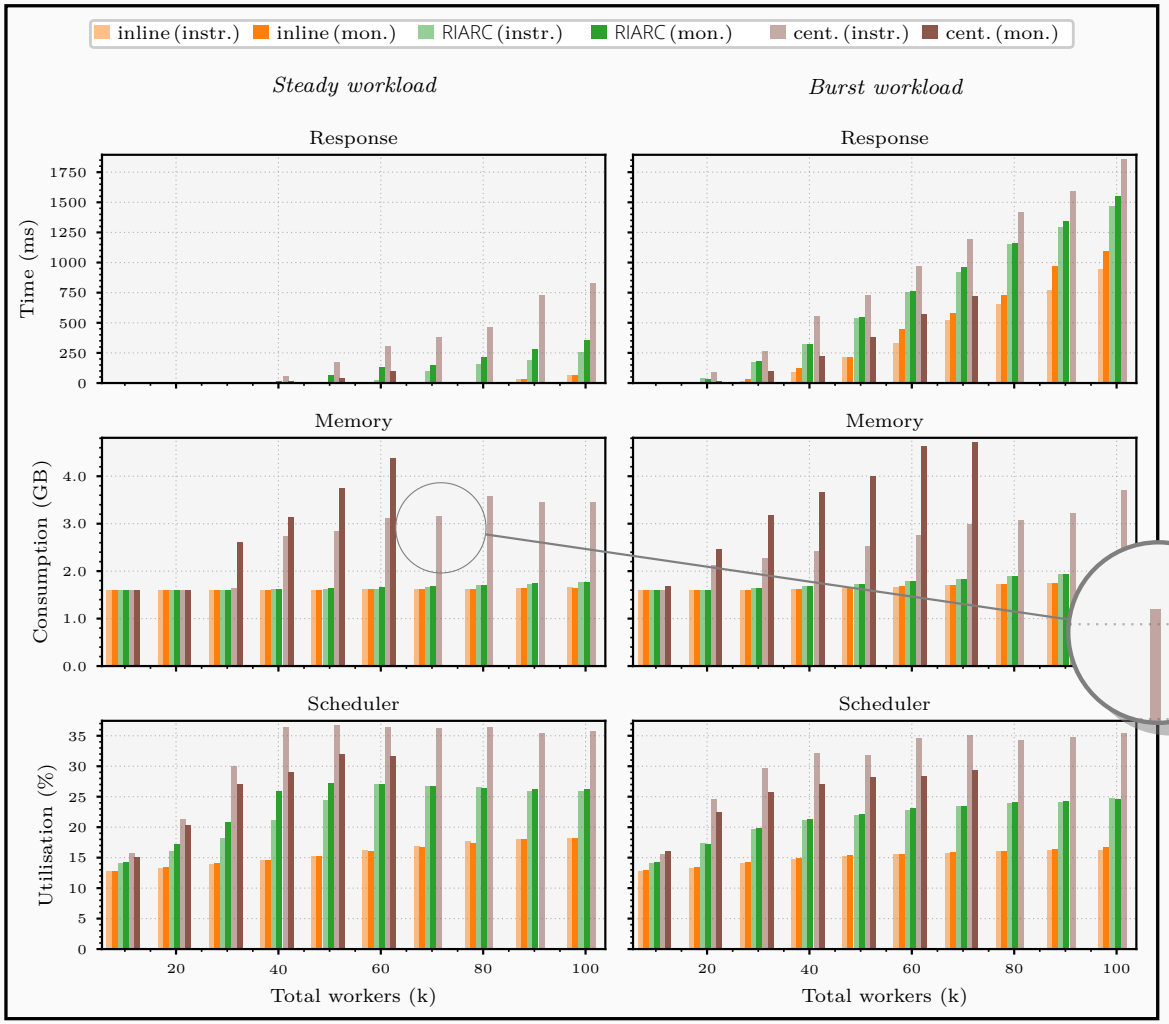
Does not **scale**

↑ memory +

↑ scheduler

= **bottleneck**

Aim: study overhead effects on the system



Instrumentation

(tracing + reporting traces soundly)

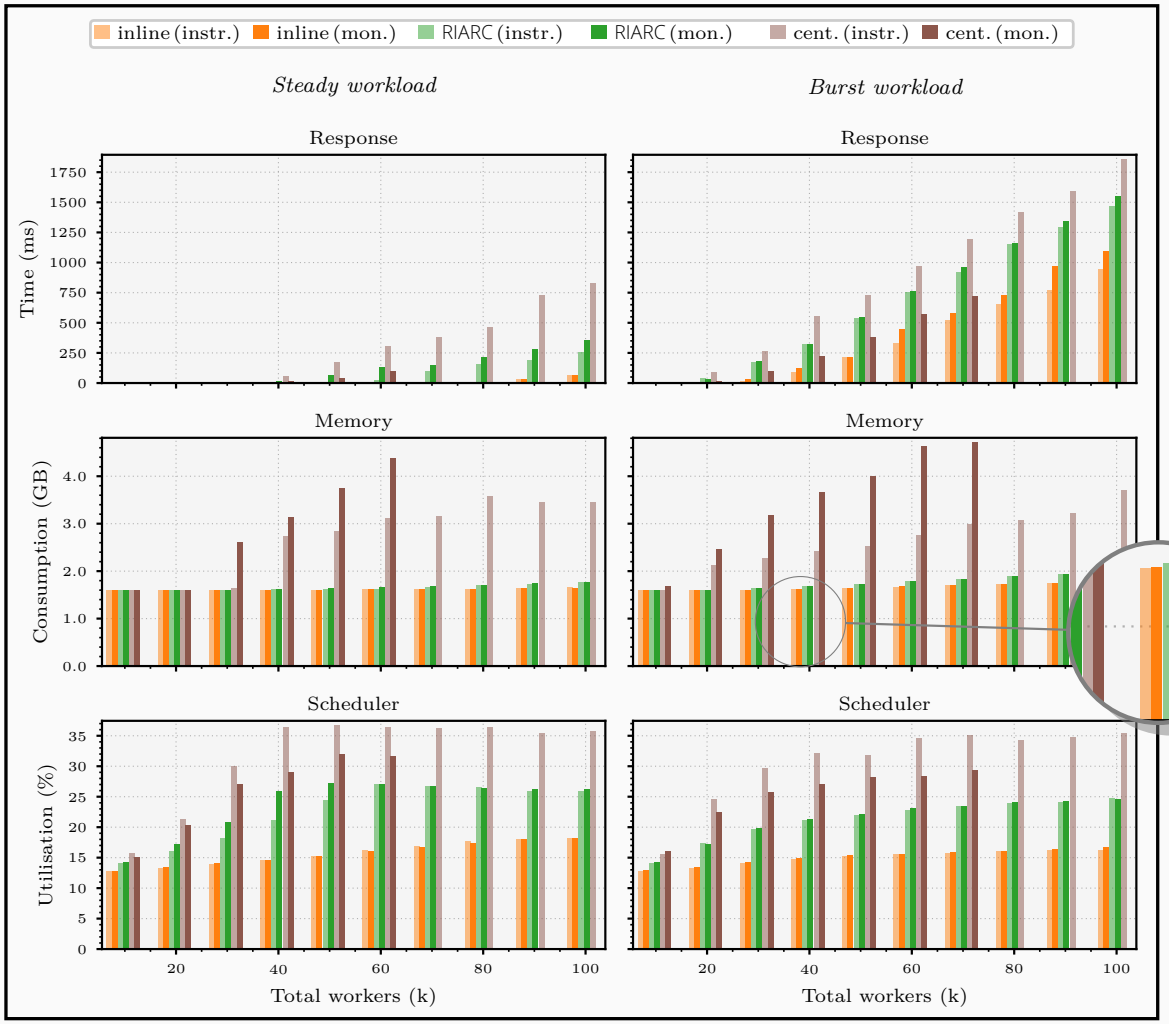
Monitoring

(instrumentation + runtime analysis)

Centralised

Crashes

Aim: study overhead effects on the system



Instrumentation

(tracing + reporting traces soundly)

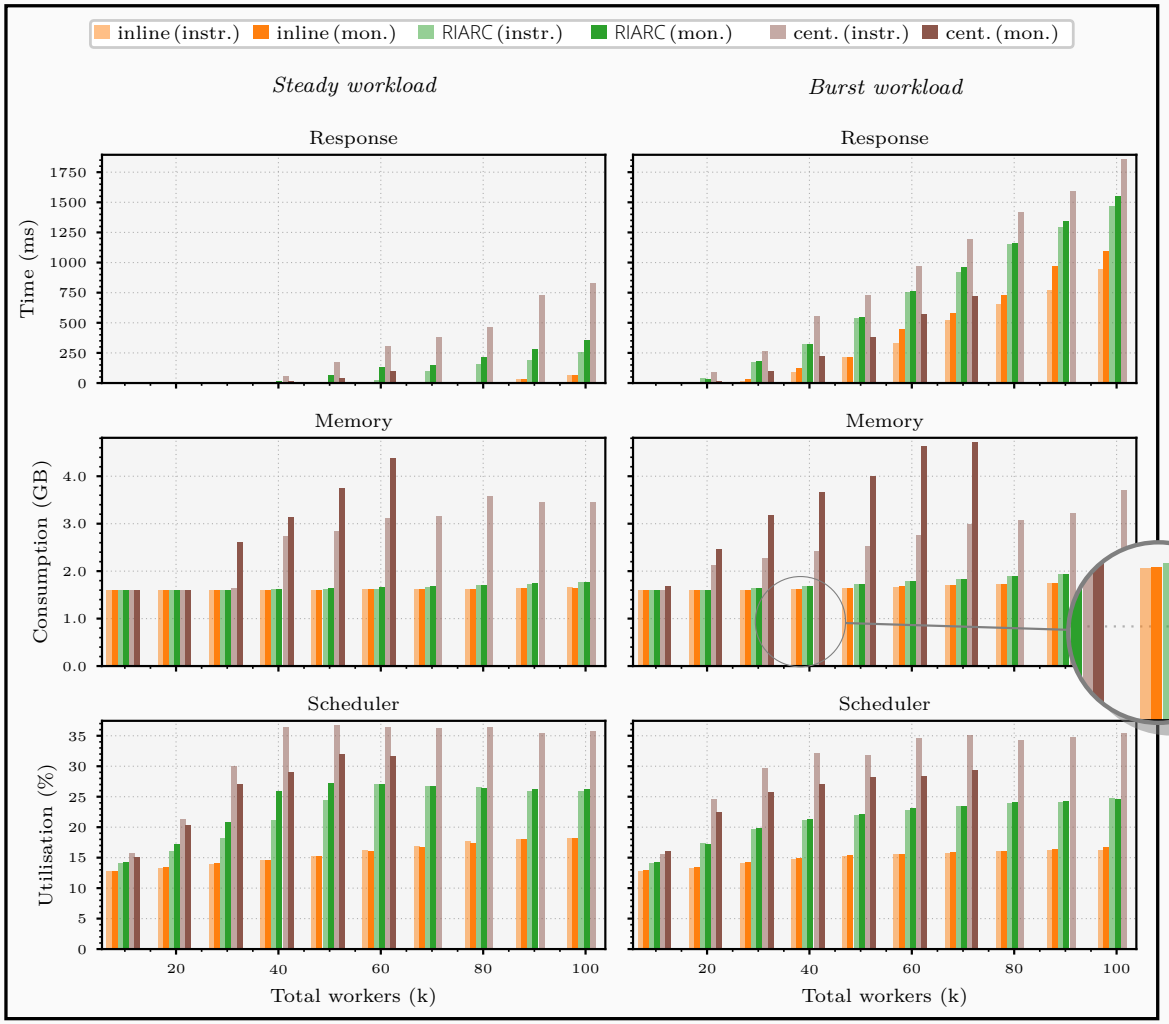
Monitoring

(instrumentation + runtime analysis)

Inline RIARC

Instrumentation carries **most** of the overhead

Aim: study overhead effects on the system



Instrumentation

(tracing + reporting traces soundly)

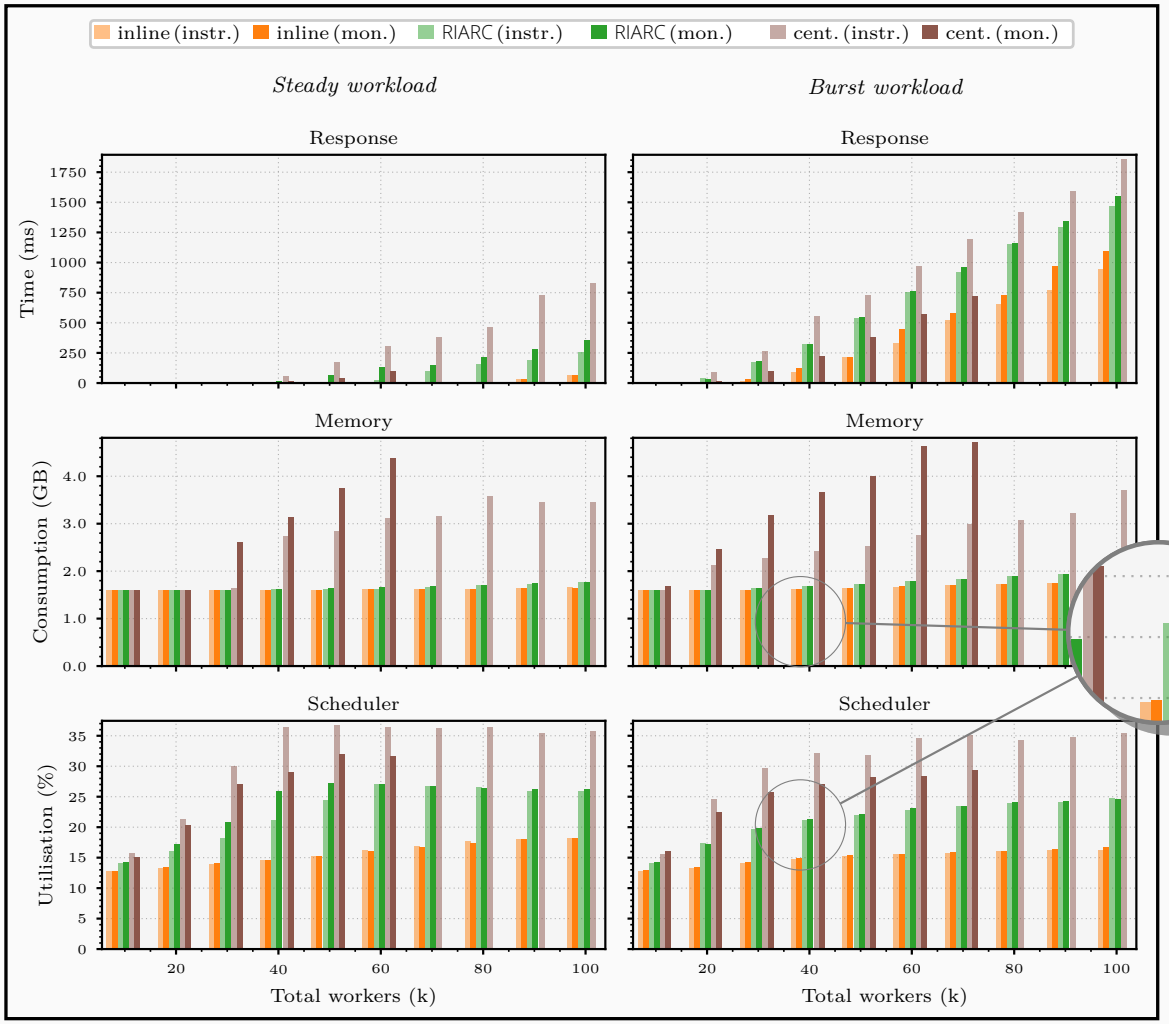
Monitoring

(instrumentation + runtime analysis)

Inline RIARC

Consume **similar** amounts of memory

Aim: study overhead effects on the system



Instrumentation

(tracing + reporting traces soundly)

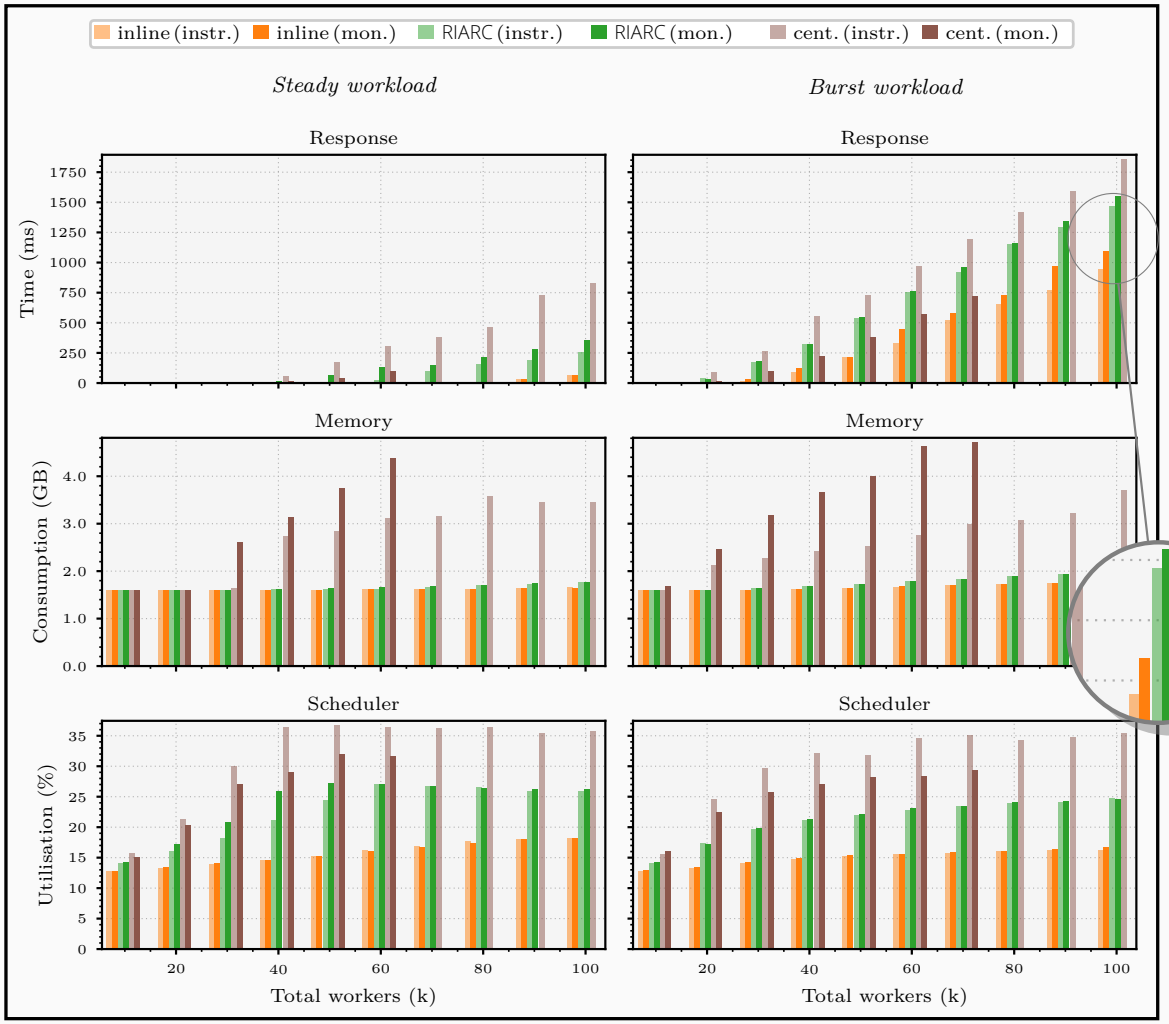
Monitoring

(instrumentation + runtime analysis)

RIARC

Uses scheduler for **dynamic reconfiguration** to keep ↓ memory = **scalable**

Aim: study overhead effects on the system



Instrumentation

(tracing + reporting traces soundly)

Monitoring

(instrumentation + runtime analysis)

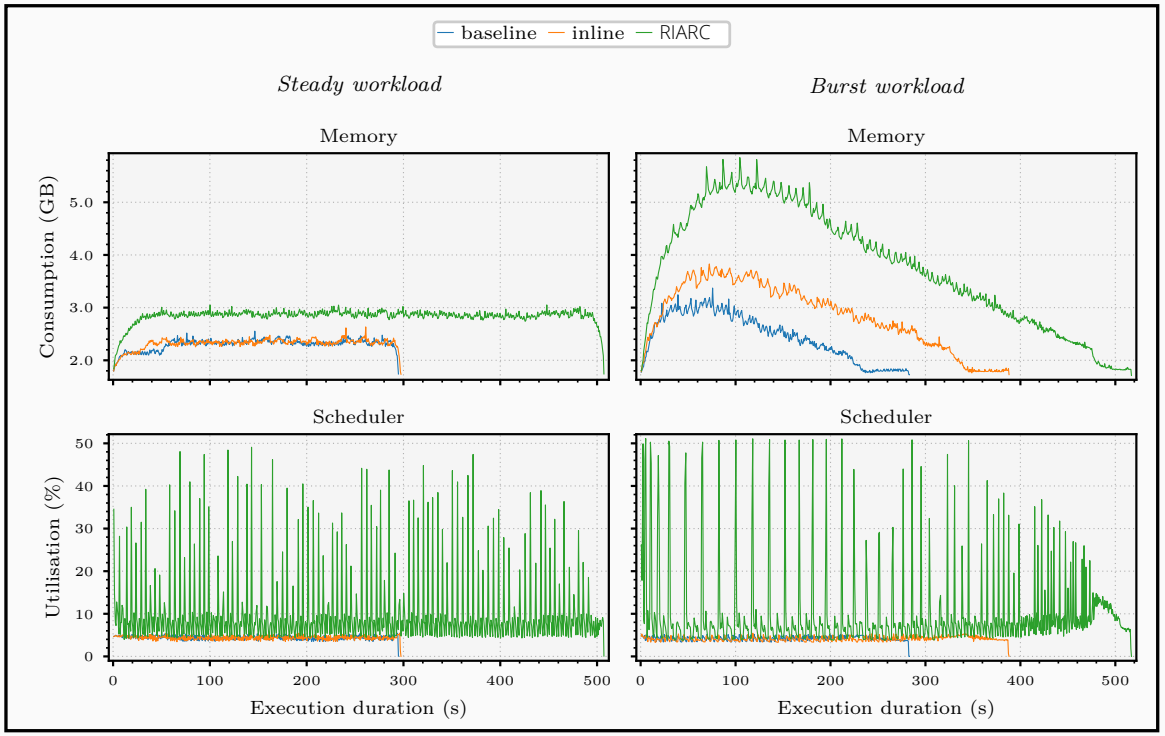
RIARC

Adds only ≈500 ms latency *vs.* inline at **max loads**

Aim: study scalability and elasticity of monitoring

Resource usage

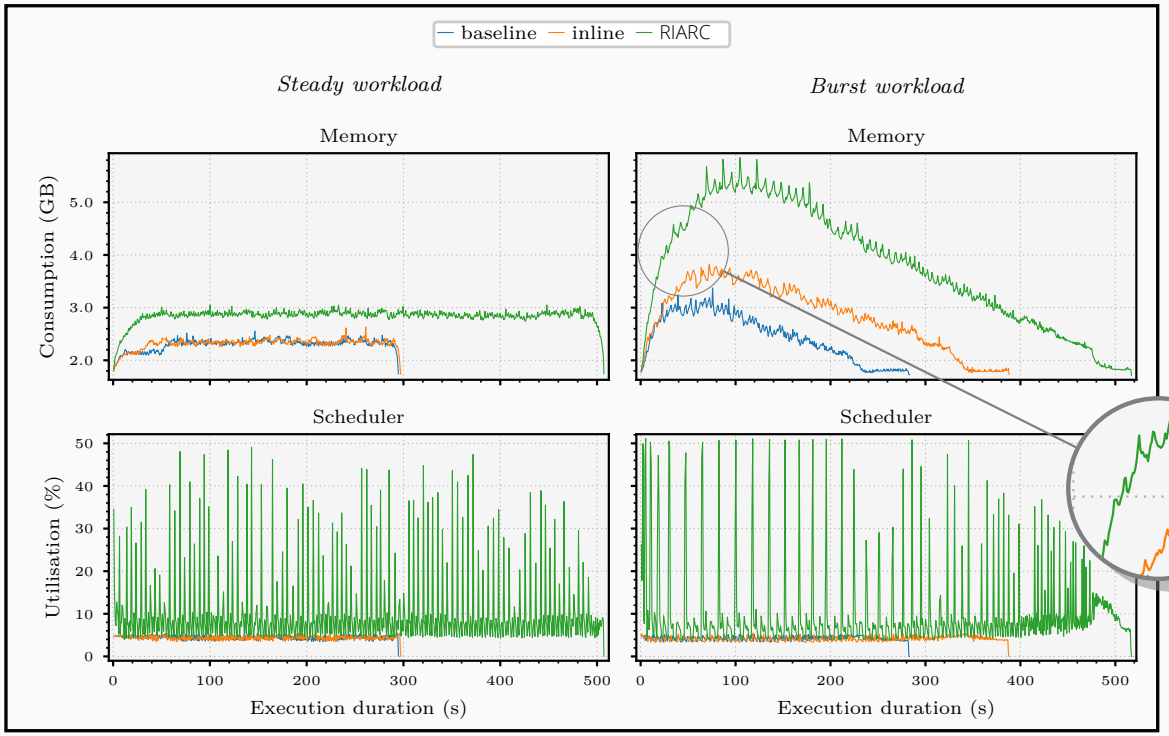
(complete system runs)



Aim: study scalability and elasticity of monitoring

Resource usage

(complete system runs)



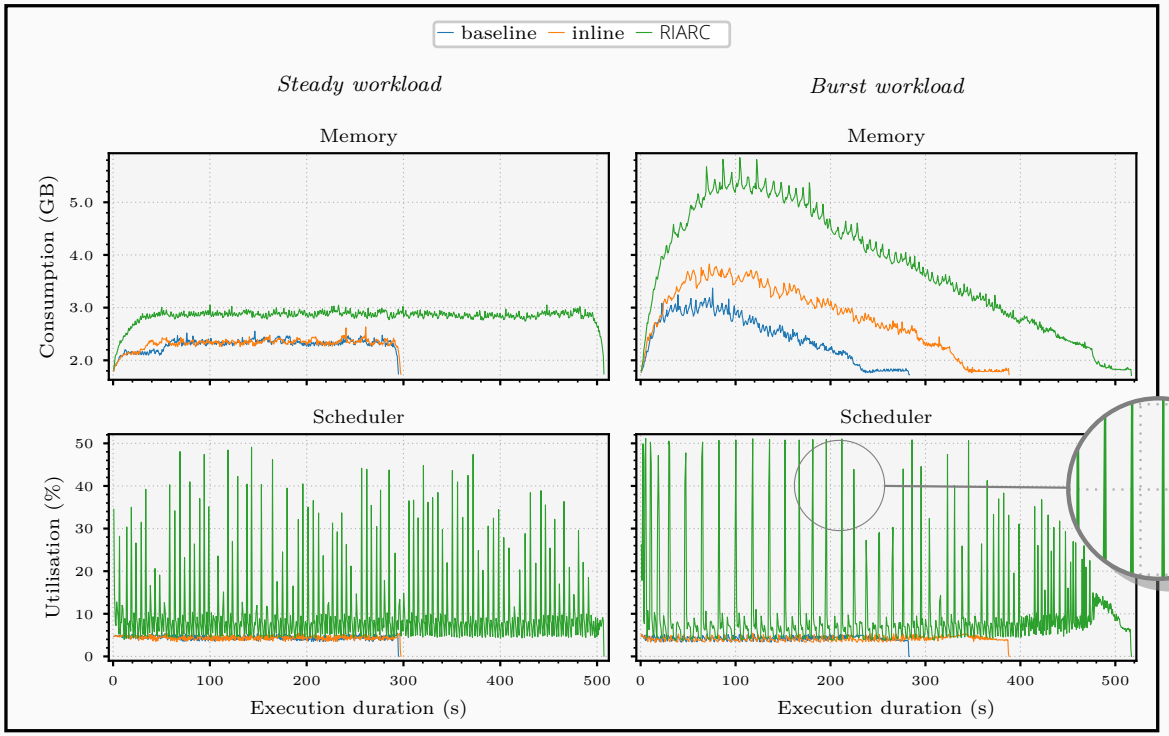
Inline RIARC

Elastic and follow shape of baseline workload

Aim: study scalability and elasticity of monitoring

Resource usage

(complete system runs)



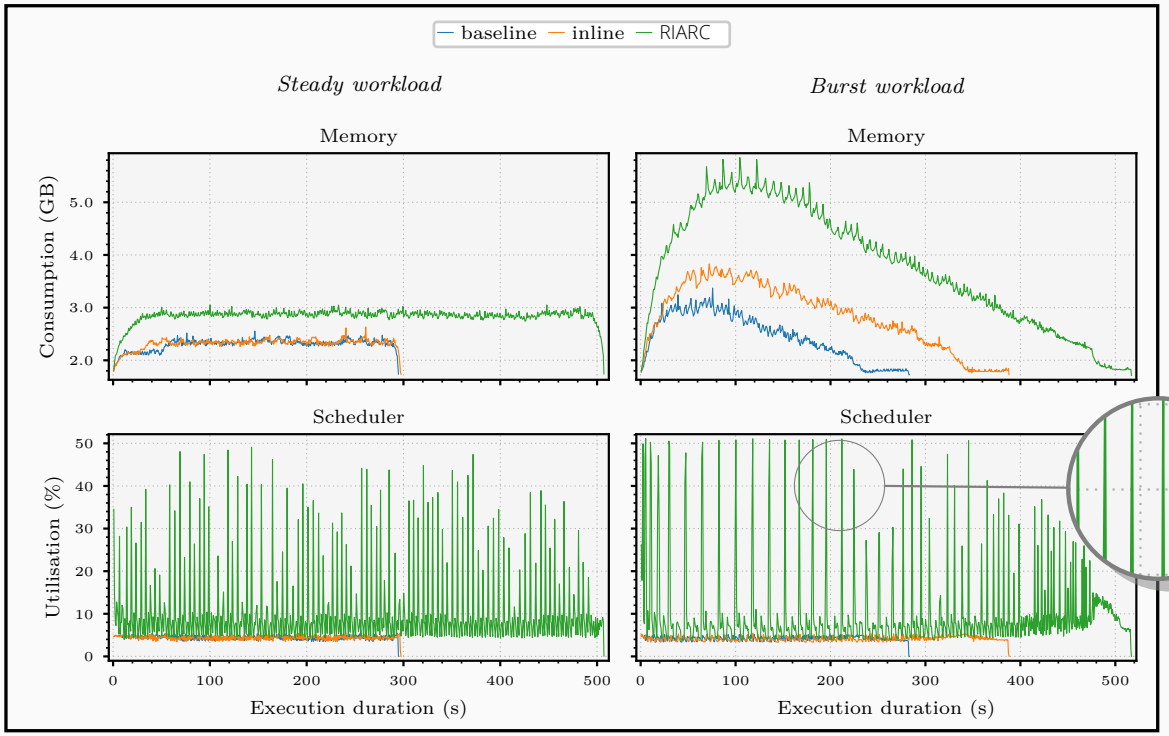
RIARC

Tracer **reconfiguration** evident in scheduler oscillations

Aim: study scalability and elasticity of monitoring

Resource usage

(complete system runs)



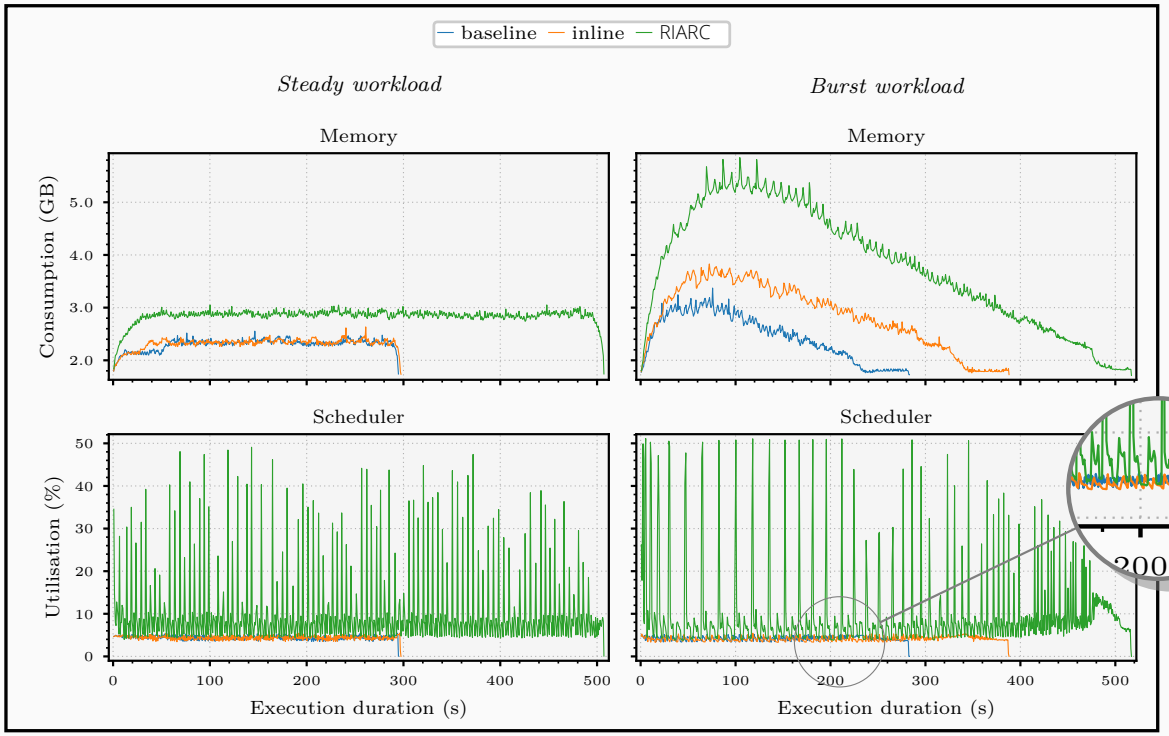
RIARC

Tracer **activation** is shown as peaks

Aim: study scalability and elasticity of monitoring

Resource usage

(complete system runs)



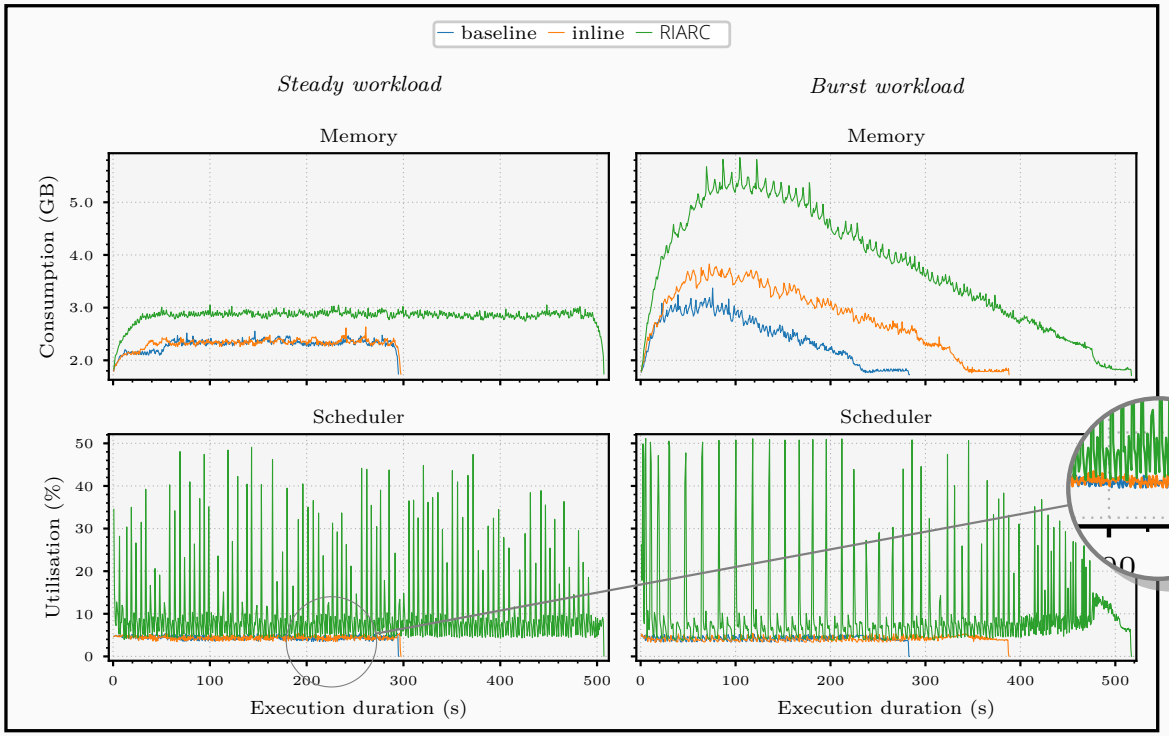
RIARC

Tracer **idle** period is shown as troughs = **message driven**

Aim: study scalability and elasticity of monitoring

Resource usage

(complete system runs)



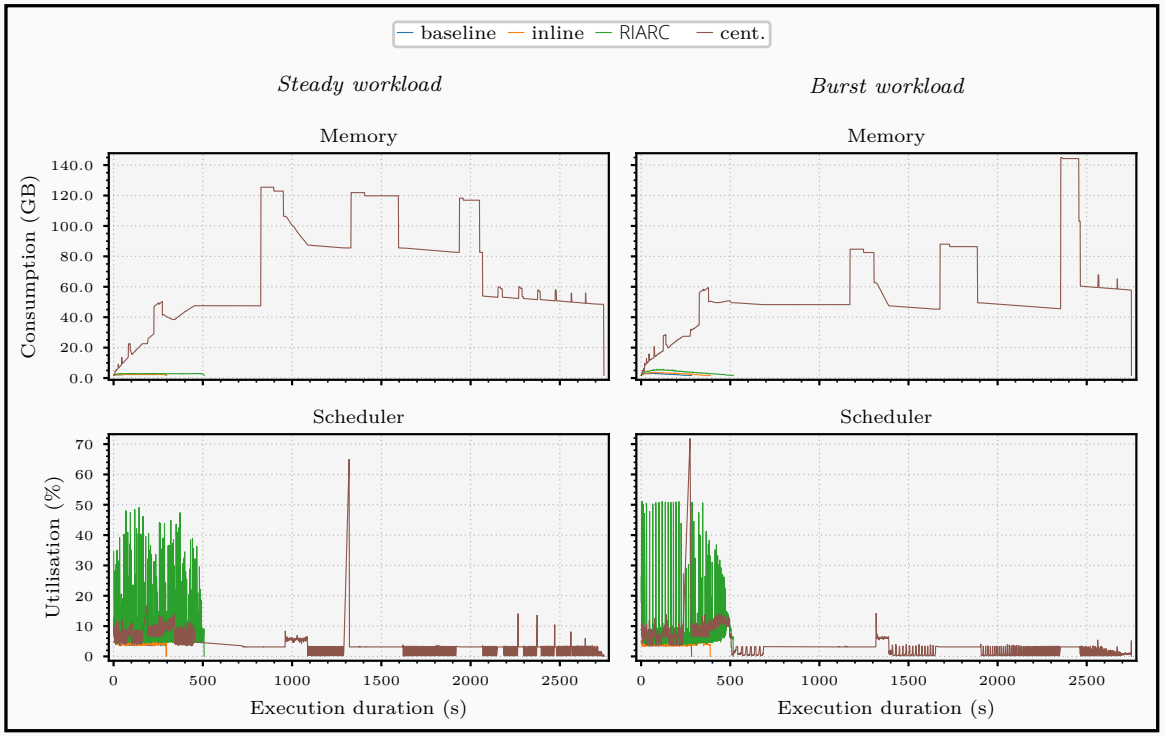
Inline

Minimal oscillations due to **lock-step** execution with system

Aim: inline and RIARC vs. centralised monitoring

Resource usage

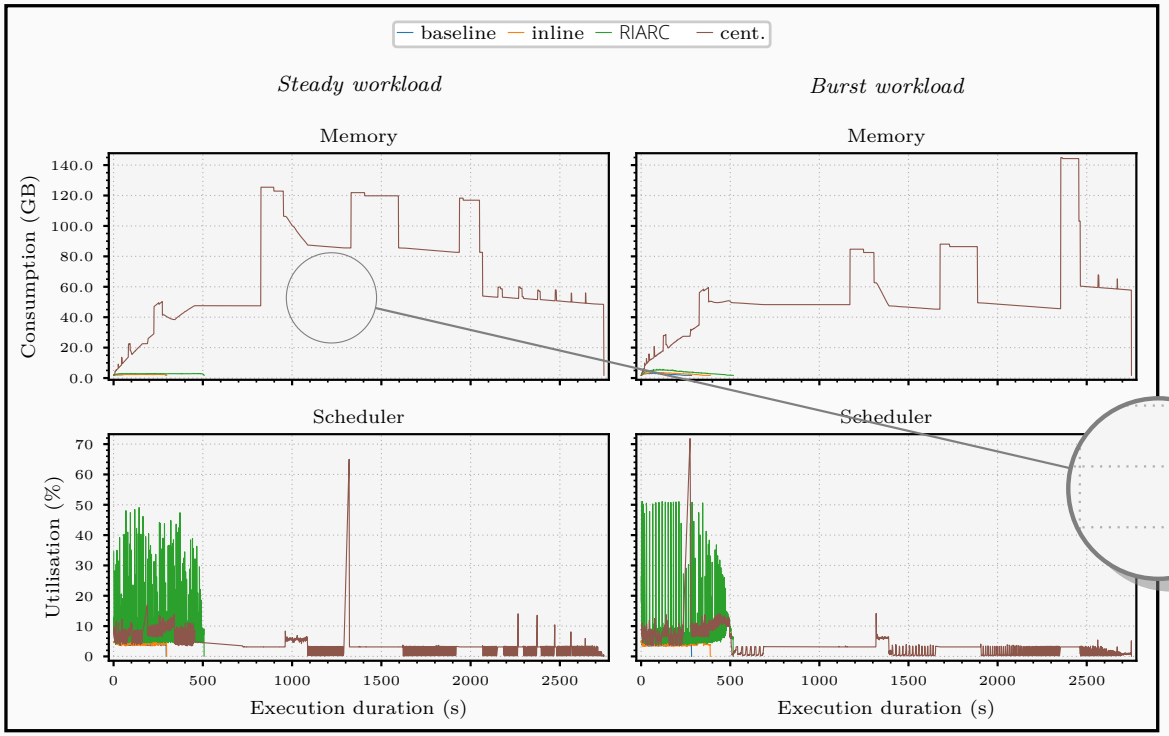
(complete system runs)



Aim: inline and RIARC vs. centralised monitoring

Resource usage

(complete system runs)



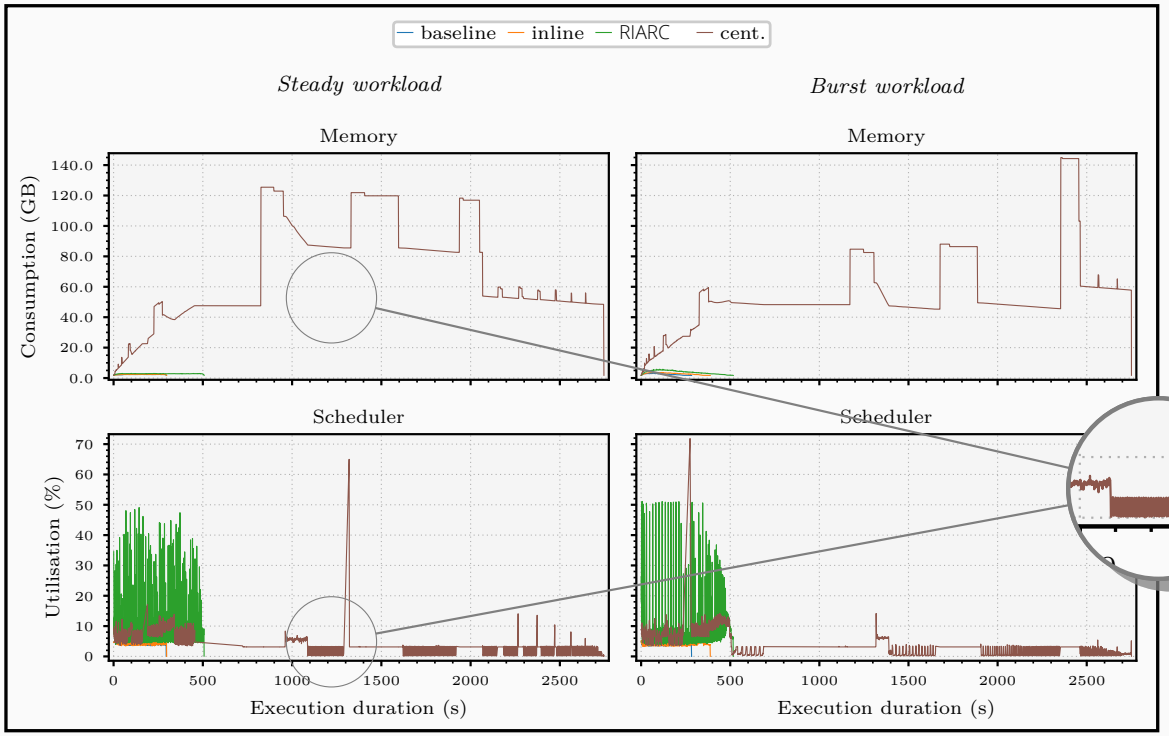
Centralised

Insensitive to workload shape

Aim: inline and RIARC vs. centralised monitoring

Resource usage

(complete system runs)



Centralised

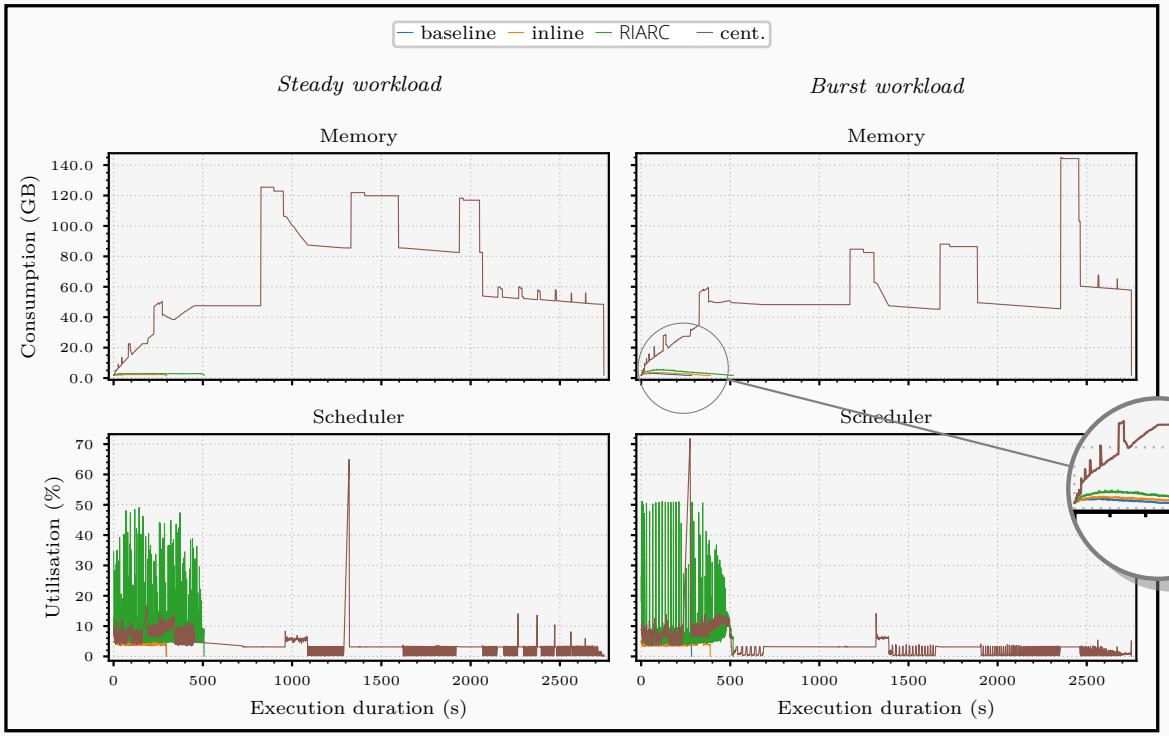
Does **not scale**

(*cf.* RIARC)

Aim: inline and RIARC vs.centralised monitoring

Resource usage

(complete system runs)



Inline RIARC

Dwarfed by memory requirements of centralised

RIARC *vs.* Inline monitoring

RIARC monitoring

- RIARC exhibits **higher** scheduler utilisation *vs.* Inline
- But ↑ scheduler utilisation **and** ↓ memory use = **scalability**
- RIARC lowers the latency **without overtaxing** the system
- RIARC leverages extra schedulers to maximise **parallelism**

Inline monitoring

- Inline lock-step execution **limits** potential parallelism gains
- Slow monitor analysis **impacted** system latency

Take away

Centralised instrumentation

Applicable ✗

Inline instrumentation

Low overhead ✓

May impact system ✗

Not always applicable ✗

RIARC

- Leaves the system **reactive** ✓
- Guarantees **trace soundness** ✓
- **Low overhead** feasible for soft real-time applications ✓

Backup slides

Aim: study how RIARC performs in typical scenarios

General-case platform to capture **commodity** hardware

High *vs.* **moderate** concurrency benchmarks

% overhead w.r.t. baseline at **maximum load**

Concurrency	Load	Response time %		Memory use %		Scheduler util %	
	200 M	Inline	RIARC	Inline	RIARC	Inline	RIARC
High	Steady	4	95	1	23	0	123
500 k	Burst	55	97	16	56	0	123
Moderate	Steady	246	194	1	8	3	52
5 k	Burst	193	190	1	10	4	50

Aim: study how RIARC performs in typical scenarios

General-case platform to capture **commodity** hardware

High *vs.* **moderate** concurrency benchmarks

% overhead w.r.t. baseline at **maximum load**

Concurrency	Load	Response time %		Memory use %		Scheduler util %	
	200 M	Inline	RIARC	Inline	RIARC	Inline	RIARC
High	Steady	4	95	1	23	0	123
500 k	Burst	55	97	16	56	0	123
Moderate	Steady	246	194	1	8	3	52
5 k	Burst	193	190	1	10	4	50

Aim: study how RIARC performs in typical scenarios

General-case platform to capture **commodity** hardware

High *vs.* **moderate** concurrency benchmarks

% overhead w.r.t. baseline at **maximum load**

Concurrency	Load	Response time %		Memory use %		Scheduler util %	
	200 M	Inline	RIARC	Inline	RIARC	Inline	RIARC
High	Steady	4	95	1	23	0	123
500 k	Burst	55	97	16	56	0	123
Moderate	Steady	246	194	1	8	3	52
5 k	Burst	193	190	1	10	4	50

Aim: study how RIARC performs in typical scenarios

General-case platform to capture **commodity** hardware

High *vs.* **moderate** concurrency benchmarks

% overhead w.r.t. baseline at **maximum load**

Concurrency	Load	Response time %		Memory use %		Scheduler util %	
	200 M	Inline	RIARC	Inline	RIARC	Inline	RIARC
High	Steady	4	95	1	23	0	123
500 k	Burst	55	97	16	56	0	123
Moderate	Steady	246	194	1	8 ↓15%	3	52
5 k	Burst	193	190	1	10	4	50

Aim: study how RIARC performs in typical scenarios

General-case platform to capture **commodity** hardware

High *vs.* **moderate** concurrency benchmarks

% overhead w.r.t. baseline at **maximum load**

Concurrency	Load	Response time %		Memory use %		Scheduler util %	
	200 M	Inline	RIARC	Inline	RIARC	Inline	RIARC
High	Steady	4	95	1	23	0	123
500 k	Burst	55	97	16	56	0	123
Moderate	Steady	246	194	1	8	3	52
5 k	Burst	193	190	1	10 ↓46%	4	50

Aim: study how RIARC performs in typical scenarios

General-case platform to capture **commodity** hardware

High *vs.* **moderate** concurrency benchmarks

% overhead w.r.t. baseline at **maximum load**

Concurrency	Load	Response time %		Memory use %		Scheduler util %	
	200 M	Inline	RIARC	Inline	RIARC	Inline	RIARC
High	Steady	4	95	1	23	0	123
500 k	Burst	55	97	16	56	0	123
Moderate	Steady	246	194	1	8	3	52 ↓71%
5 k	Burst	193	190	1	10	4	50, ↓73%

Aim: study how RIARC performs in typical scenarios

General-case platform to capture **commodity** hardware

High *vs.* **moderate** concurrency benchmarks

% overhead w.r.t. baseline at **maximum load**

Concurrency	Load	Response time %		Memory use %		Scheduler util %	
	200 M	Inline	RIARC	Inline	RIARC	Inline	RIARC
High	Steady	4	95	1	23	0	123
500 k	Burst	55	97	16	56	0	123
Moderate	Steady	246	194 ↑99%	1	8	3	52
5 k	Burst	193	190	1	10	4	50

Aim: study how RIARC performs in typical scenarios

General-case platform to capture **commodity** hardware

High *vs.* **moderate** concurrency benchmarks

% overhead w.r.t. baseline at **maximum load**

Concurrency	Load	Response time %		Memory use %		Scheduler util %	
	200 M	Inline	RIARC	Inline	RIARC	Inline	RIARC
High	Steady	4	95	1	23	0	123
500 k	Burst	55	97	16	56	0	123
Moderate	Steady	246	194	1	8	3	52
5 k	Burst	193	190 ↑93%	1	10	4	50

Aim: study how RIARC performs in typical scenarios

General-case platform to capture **commodity** hardware

High *vs.* **moderate** concurrency benchmarks

% overhead w.r.t. baseline at **maximum load**

Concurrency	Load	Response time %		Memory use %		Scheduler util %	
	200 M	Inline	RIARC	Inline	RIARC	Inline	RIARC
High	Steady	4	95	1	23	0	123
500 k	Burst	55	97	16	56	0	123
Moderate	Steady	246 ↑242%	194	1	8	3	52
5 k	Burst	193	190	1	10	4	50

Aim: study how RIARC performs in typical scenarios

General-case platform to capture **commodity** hardware

High *vs.* **moderate** concurrency benchmarks

% overhead w.r.t. baseline at **maximum load**

Concurrency	Load	Response time %		Memory use %		Scheduler util %	
	200 M	Inline	RIARC	Inline	RIARC	Inline	RIARC
High	Steady	4	95	1	23	0	123
500 k	Burst	55	97	16	56	0	123
Moderate	Steady	246	194	1	8	3	52
5 k	Burst	193 ↑138%	190	1	10	4	50

Aim: study how RIARC performs in typical scenarios

General-case platform to capture **commodity** hardware

High *vs.* **moderate** concurrency benchmarks

% overhead w.r.t. baseline at **maximum load**

Concurrency	Load	Response time %		Memory use %		Scheduler util %	
	200 M	Inline	RIARC	Inline	RIARC	Inline	RIARC
High	Steady	4	95	1	23	0	123
500 k	Burst	55	97	16	56	0	123
Moderate	Steady	246	194	1	8	3	52
5 k	Burst	193	190	1	10	4	50

Response time Inline *vs.* RIARC: 116 ms *vs.* 98 ms (Steady) and 182 ms *vs.* 179 ms (Burst)