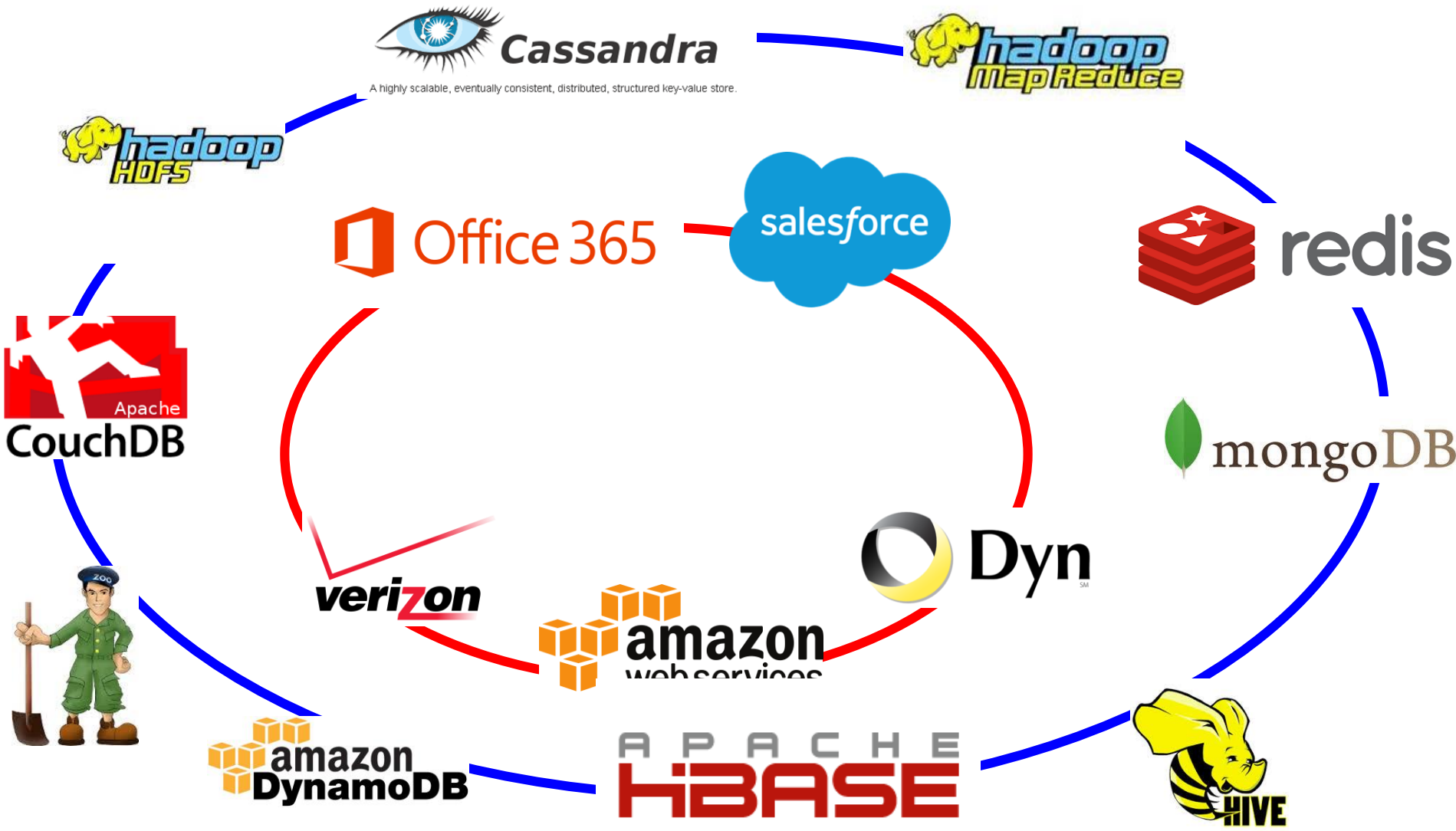
DCatch: Automatically Detecting Distributed Concurrency Bugs in Cloud Systems

Haopeng Liu, Guangpu Li, Jeffrey Lukman, Jiaxin Li, Shan Lu, Haryadi Gunawi, and Chen Tian*

THE UNIVERSITY OF CHICAGO

*

Cloud systems



Cloud systems

Cassandra

A highly scalable, eventually consistent, distributed, structured key-value store.

hadoop MapReduce

hadoop HDFS

The 10 Biggest Cloud Outages of 2016

redis

www.crn.com/slide-shows/cloud/.../the-10-bigges

Apache CouchDB

ongoDB

Dec 29, 2016 - The **biggest** cloud **outages** of **2016** incl

large, are increasingly vulnerable from **downtime**.

amazon DynamoDB

APACHE HBASE

HIVE

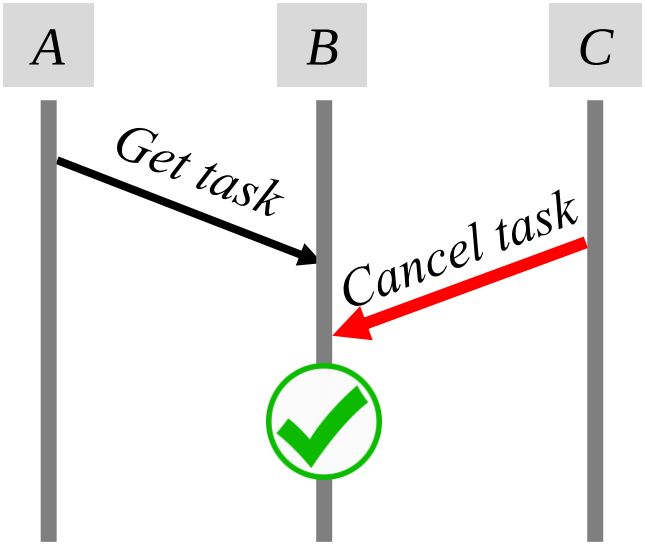
Distributed concurrency bugs (DCbugs)

Distributed concurrency bugs (DCbugs)

- **Unexpected timing** among **distributed operations**

Distributed concurrency bugs (DCbugs)

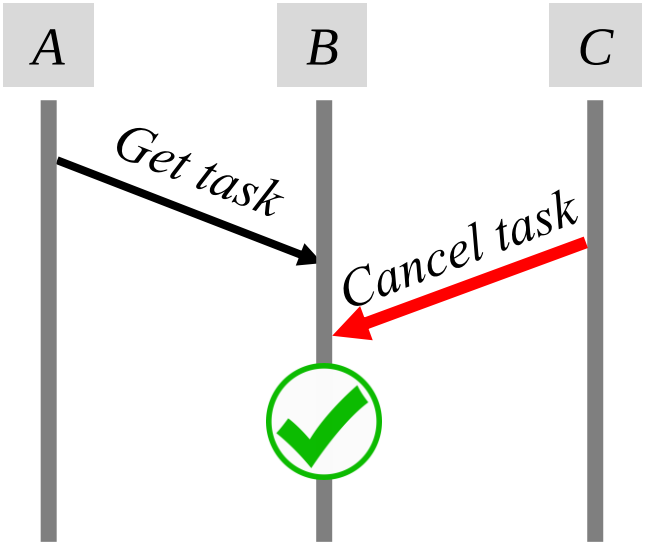
- Unexpected timing among distributed operations
- Example

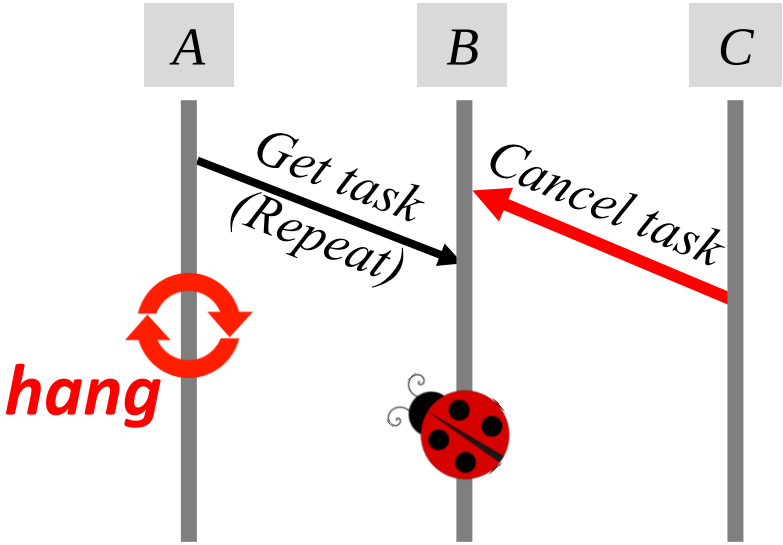


MapReduce-3274

Distributed concurrency bugs (DCbugs)

- Unexpected timing among distributed operations
- Example





MapReduce-3274

DCbugs need to be tackled

- Common in distributed systems [1, 2, 3]
  - 26% failures caused by non-deterministic [1]
  - 6% software bugs in clouds system [2]

[1] Yuan. Simple Testing Can Prevent Most Critical Failures. In OSDI'14
[2] Gunawi. What Bugs Live in the Cloud?. In SoCC'14
[3] Leesatapornwongsa. TaxDC. In ASPLOS'16

DCbugs need to be tackled

- Common in distributed systems [1, 2, 3]
- Difficult to avoid, expose and diagnose

[1] Yuan. Simple Testing Can Prevent Most Critical Failures. In OSDI'14
[2] Gunawi. What Bugs Live in the Cloud?. In SoCC'14
[3] Leesatapornwongsa. TaxDC. In ASPLOS'16

DCbugs need to be tackled

- Common in distributed systems [1, 2, 3]
- Difficult to avoid, expose and diagnose

Hadoop Map/Reduce / MAPREDUCE-3274

"That is one monster of a race!"

[1] Yuan. Simple Testing Can Prevent Most Critical Failures. In OSDI'14
[2] Gunawi. What Bugs Live in the Cloud?. In SoCC'14
[3] Leesatapornwongsa. TaxDC. In ASPLOS'16

DCbugs need to be tackled

- Common in distributed systems [1, 2, 3]
- Difficult to avoid, expose and diagnose

Hadoop Map/Reduce / MAPREDUCE-3274

HBase / HBASE-4397

"There isn't a week going by without new bugs about races."

[1] Yuan. Simple Testing Can Prevent Most Critical Failures. In OSDI'14
[2] Gunawi. What Bugs Live in the Cloud?. In SoCC'14
[3] Leesatapornwongsa. TaxDC. In ASPLOS'16

DCbugs need to be tackled

- Common in distributed systems [1, 2, 3]
- Difficult to avoid, expose and diagnose

Hadoop Map/Reduce / MAPREDUCE-3274

HBase / HBASE-4397

HBase / HBASE-6147

"We have already fix many cases,
however it seems exist many other [racing] cases."

[1] Yuan. Simple Testing Can Prevent Most Critical Failures. In OSDI'14
[2] Gunawi. What Bugs Live in the Cloud?. In SoCC'14
[3] Leesatapornwongsa. TaxDC. In ASPLOS'16

DCbugs need to be tackled

- Common in distributed systems [1, 2, 3]
- Difficult to avoid, expose and diagnose

Hadoop Map/Reduce / MAPREDUCE-3274

HBase / HBASE-4397

HBase / HBASE-6147

Hadoop Map/Reduce / MAPREDUCE-4819

"This has become quite messy, **sigh.**"

[1] Yuan. Simple Testing Can Prevent Most Critical Failures. In OSDI'14
[2] Gunawi. What Bugs Live in the Cloud?. In SoCC'14
[3] Leesatapornwongsa. TaxDC. In ASPLOS'16

DCbugs need to be tackled

- Common in distributed systems [1, 2, 3]
- Difficult to avoid, expose and diagnose

Hadoop Map/Reduce / MAPREDUCE-3274

HBase / HBASE-4397

HBase / HBASE-6147

Hadoop Map/Reduce / MAPREDUCE-4819

Hadoop Map/Reduce / MAPREDUCE-4099

"Great catch, Sid! Apologies for missing the race condition."

[1] Yuan. Simple Testing Can Prevent Most Critical Failures. In OSDI'14
[2] Gunawi. What Bugs Live in the Cloud?. In SoCC'14
[3] Leesatapornwongsa. TaxDC. In ASPLOS'16

DCbugs need to be tackled

- Common in distributed systems [1, 2, 3]
- Difficult to avoid, expose and diagnose

Hadoop Map/Reduce / MAPREDUCE-3274

HBase / HBASE-4397

HBase / HBASE-6147

Hadoop Map/Reduce / MAPREDUCE-4819

Hadoop Map/Reduce / MAPREDUCE-4099

Hadoop Map/Reduce / MAPREDUCE-3634

"We [prefer] debug crashes instead of hanging jobs."

[1] Yuan. Simple Testing
[2] Gunawi. What Bugs Live in the Cloud?. In SoCC'14
[3] Leesatapornwongsa. TaxDC. In ASPLOS'16

DCbugs need to be tackled

- Common in distributed systems [1, 2, 3]
- Difficult to avoid, expose and diagnose

Hadoop Map/Reduce / MAPREDUCE-3274

HBase / HBASE-4397

HBase / HBASE-6147

Can we detect DCbugs before they manifest?

Hadoop Map/Reduce / MAPREDUCE-3634

"We [prefer] debug crashes instead of hanging jobs."

[1] Yuan. Simple Testing
[2] Gunawi. What Bugs Live in the Cloud?. In SoCC'14
[3] Leesatapornwongsa. TaxDC. In ASPLOS'16

Previous work

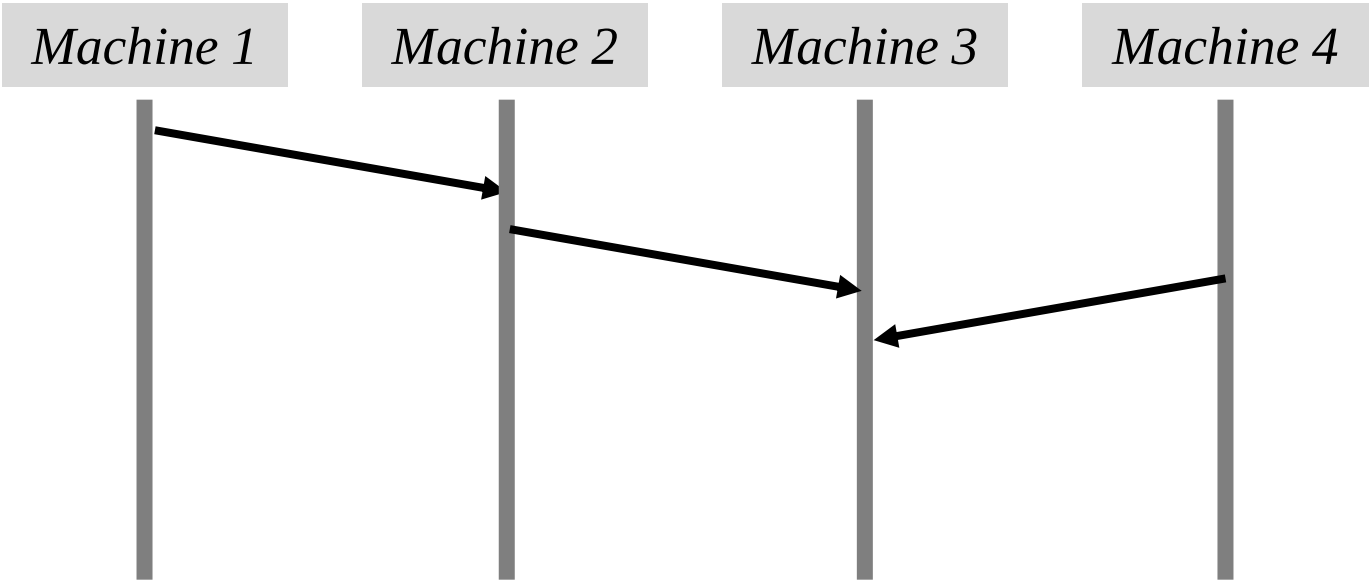
- Model checking
  - Work on abstracted models
  - Face state-space explosion issue

Our idea

- Follow the philosophy of traditional concurrency bug detection

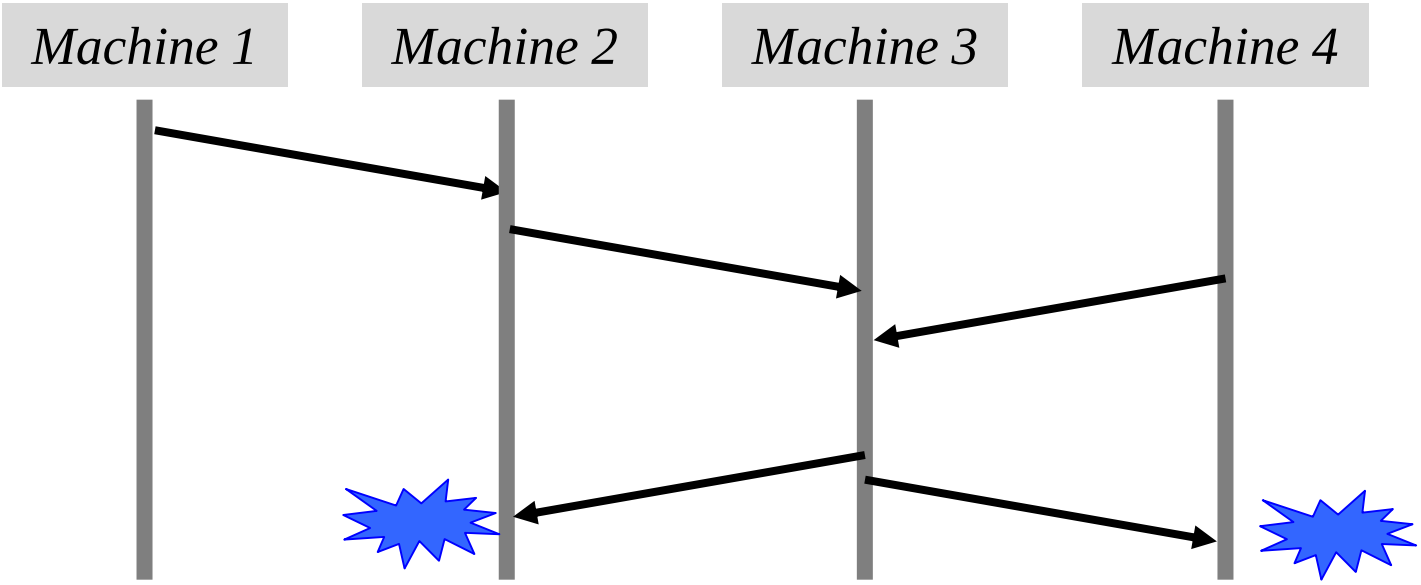
Our idea

- Follow the philosophy of traditional concurrency bug detection



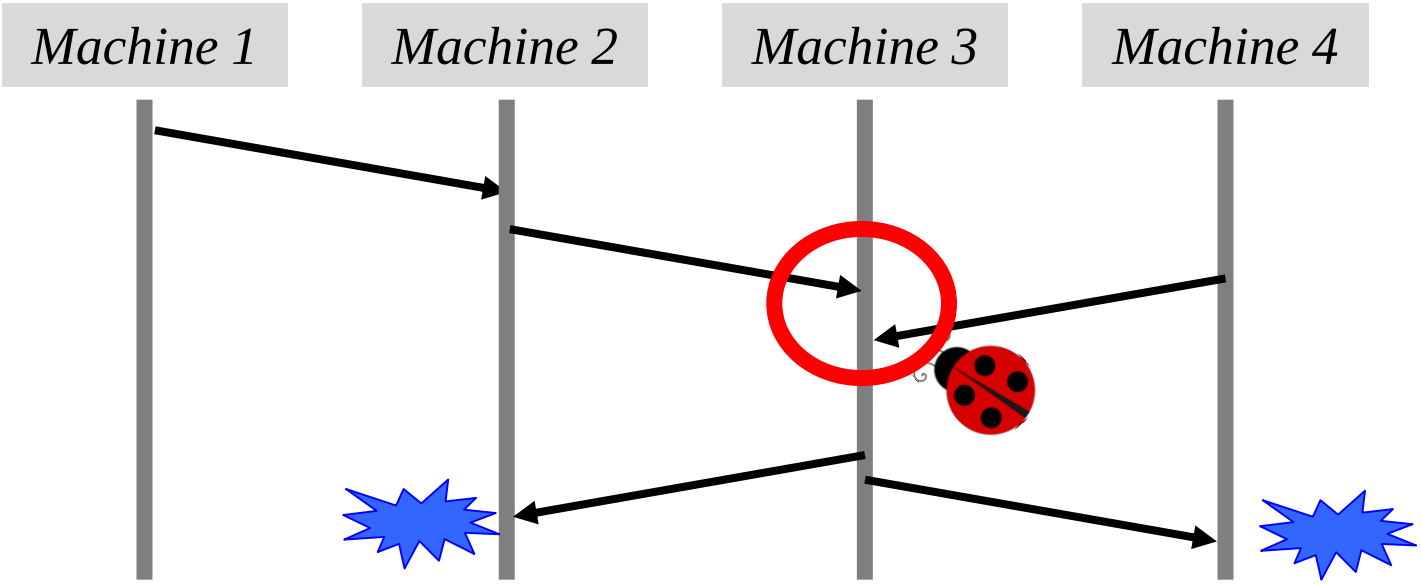
Our idea

- Follow the philosophy of traditional concurrency bug detection

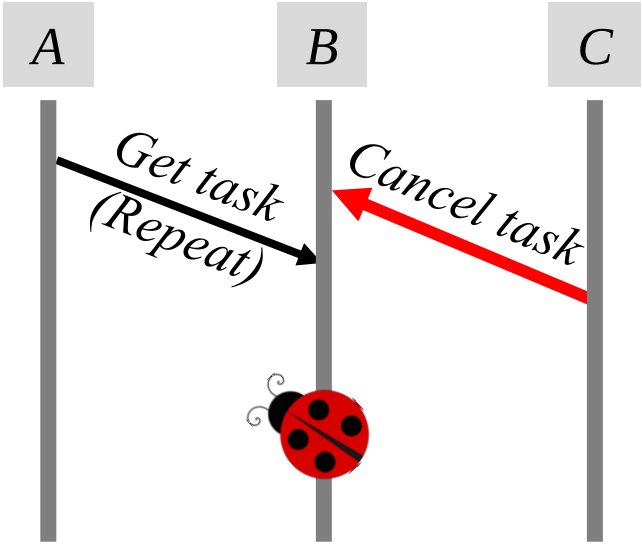


Our idea

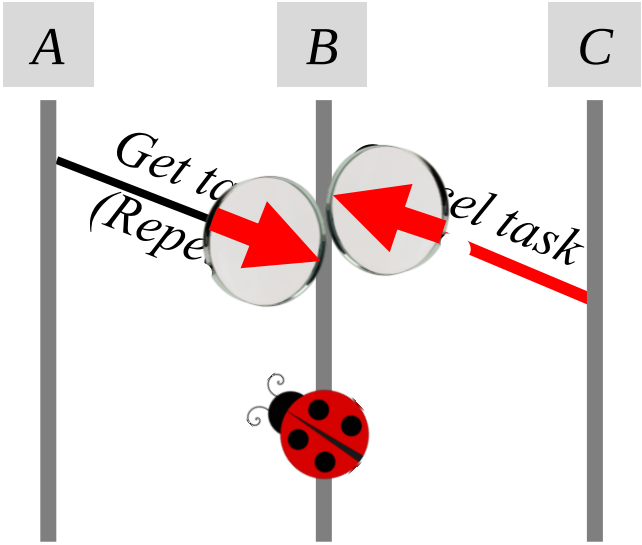
- Follow the philosophy of traditional concurrency bug detection



Example



Example



B

```
//RPC thread
Task getTask(jID){
   ...
   return jMap.get(jID);
}
```

```
//UnReg thread
void unReg(jID){
   jMap.remove(jID);
   ....
}
```

Local concurrency bug detection

ASPLOS

ASPLOS Architectural Support for Programming

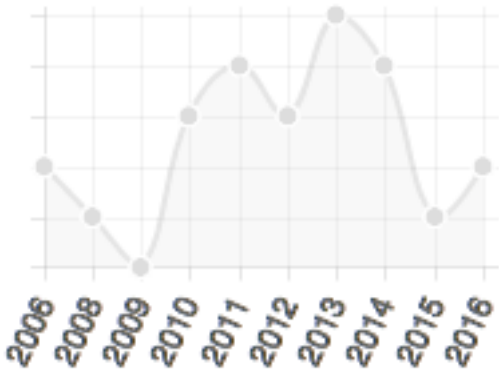
ASPLOS is a multi-disciplinary conference for rese
compilers, languages, operating systems, networ
and engineers to present their latest research fin
computer systems innovations of the past two de
multiprocessors, clusters and networks-of-workst

This conference occurs at a time when computer
processor performance scaling and to new demar
increasingly important as boundaries between ha
capabilities of computing devices beco expand

Search within ASPLOS: "concurrency bug" "race"

35 results found

Refine by Publication Year



Published Since 2006

Local concurrency bug detection

ASPLOS

ASPLOS Architectural Support for Programming

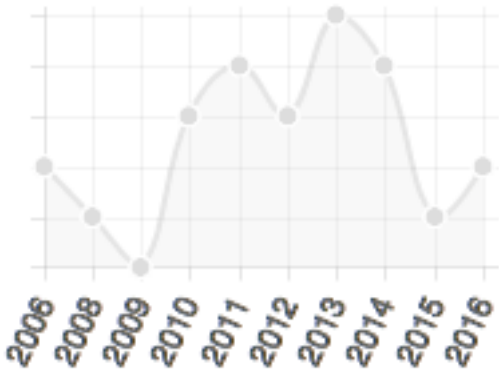
ASPLOS is a multi-disciplinary conference for rese
compilers, languages, operating systems, networ
and engineers to present their latest research fin
computer systems innovations of the past two de
multiprocessors, clusters and networks-of-workst

This conference occurs at a time when computer
processor performance scaling and to new demar
increasingly important as boundaries between ha
capabilities of computing devices beco expand

Search within ASPLOS: "concurrency bug" "race"

35 results found

Refine by Publication Year



Published Since 2006

Local concurrency bug detection

ASPLOS

ASPLOS Architectural Support for Programming

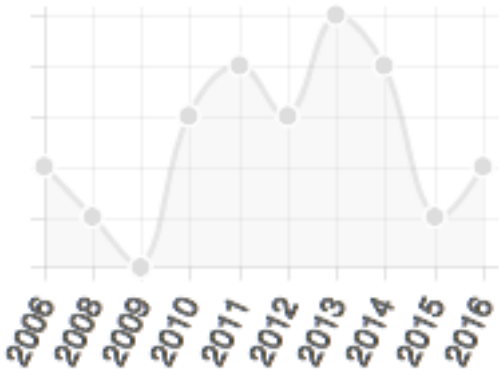
ASPLOS is a multi-disciplinary conference for rese
compilers, languages, operating systems, networ
and engineers to present their latest research fin
computer systems innovations of the past two de
multiprocessors, clusters and networks-of-workst

This conference occurs at a time when computer
processor performance scaling and to new demar
increasingly important as boundaries between ha
capabilities of computing devices beco expand

Search within ASPLOS: "concurrency bug" "race"

35 results found

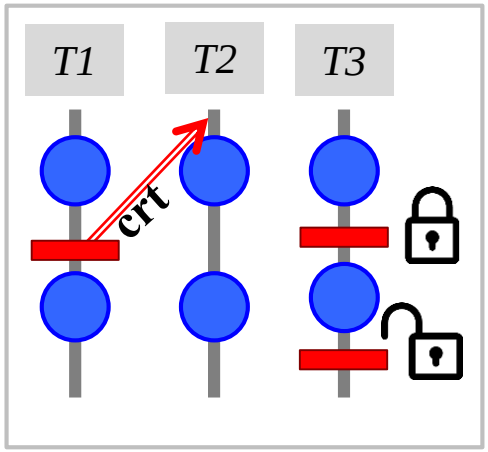
Refine by Publication Year



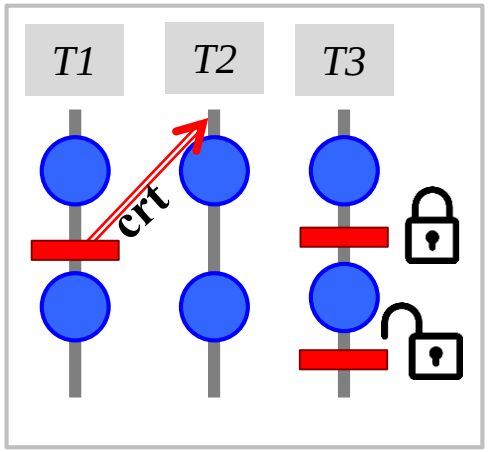
Published Since 2006

Is the problem solved?

Local concurrency bug detection



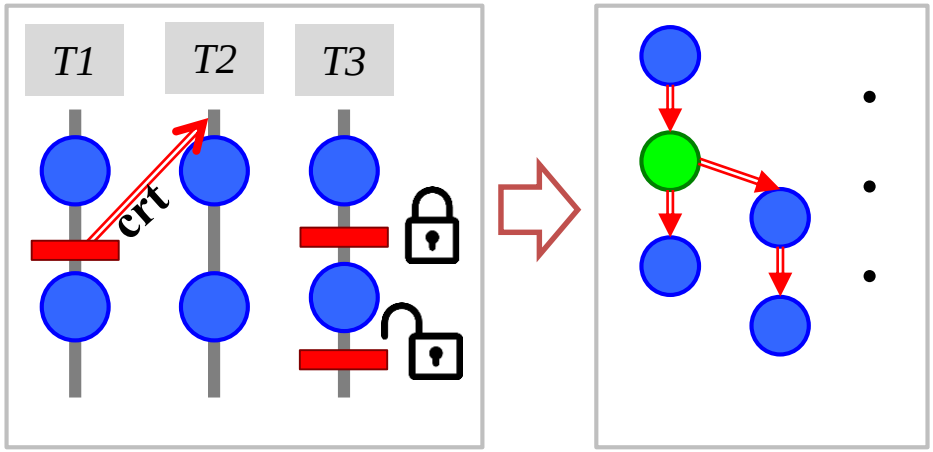
Local concurrency bug detection

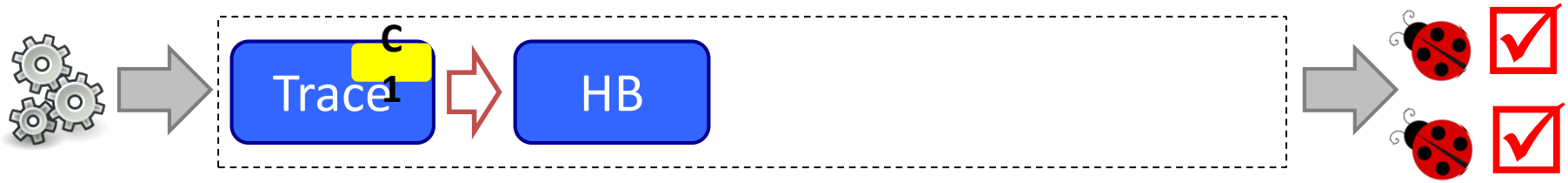


C1: How to handle the huge amount of mem accesses?

Challenges

Local concurrency bug detection

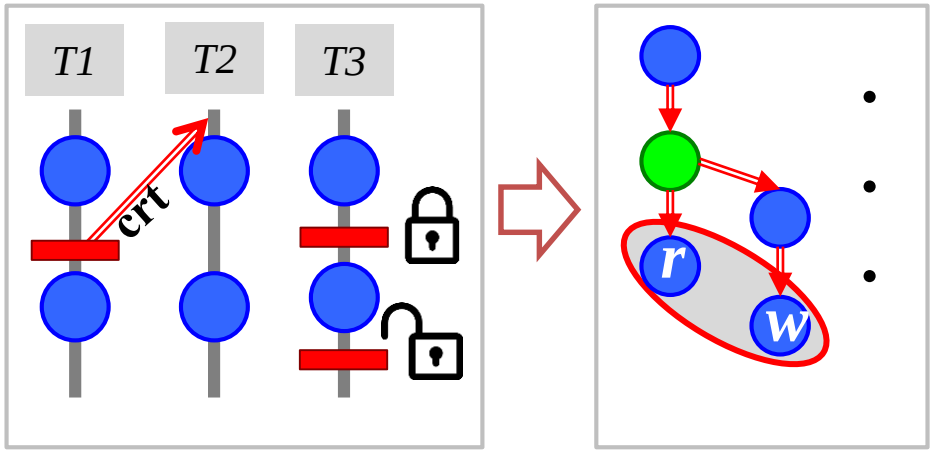


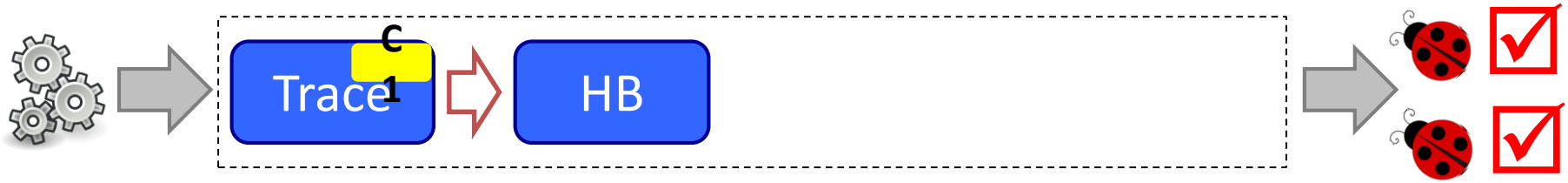


C1: How to handle the huge amount of mem accesses?

Challenges

Local concurrency bug detection

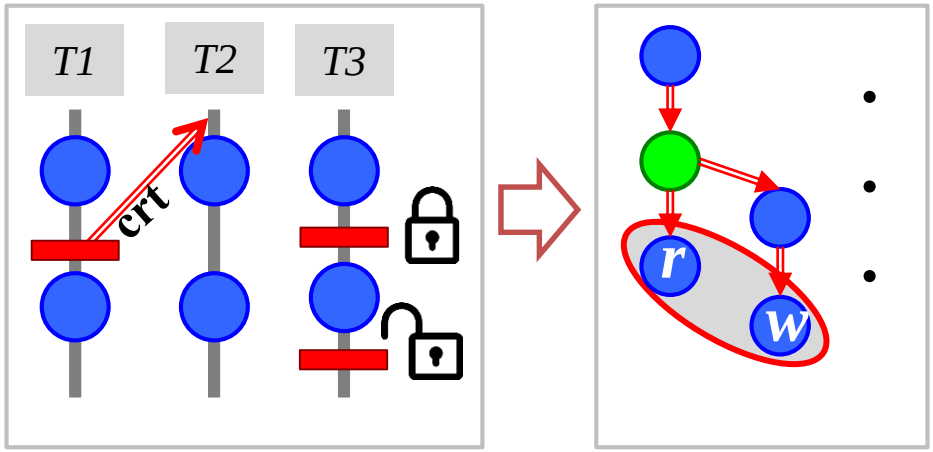


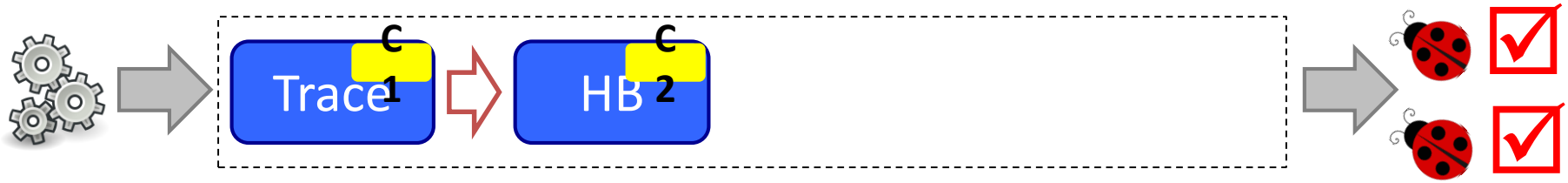


C1: How to handle the huge amount of mem accesses?

Challenges

Local concurrency bug detection



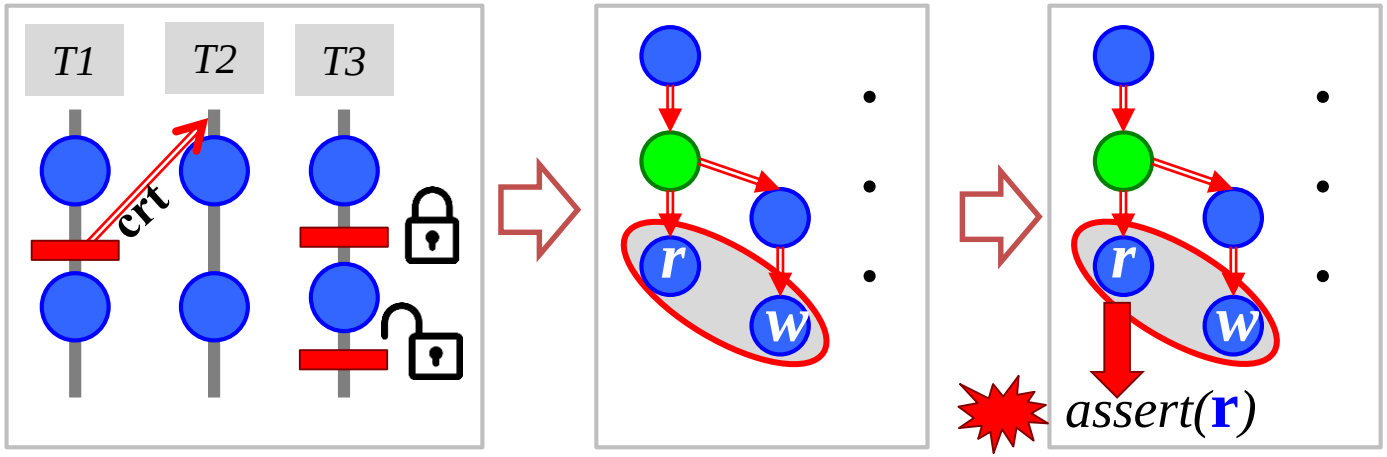


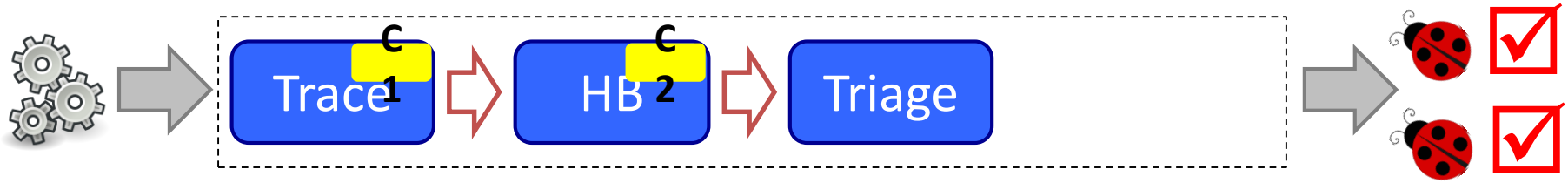
C1: How to handle the huge amount of mem accesses?

C2: What's the happens-before model?

Challenges

Local concurrency bug detection



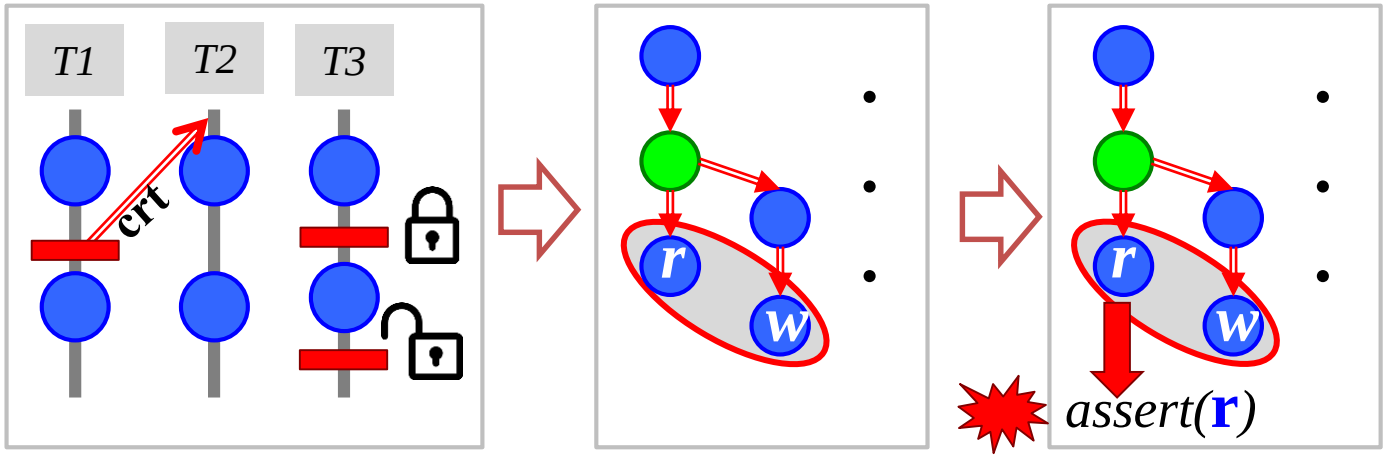


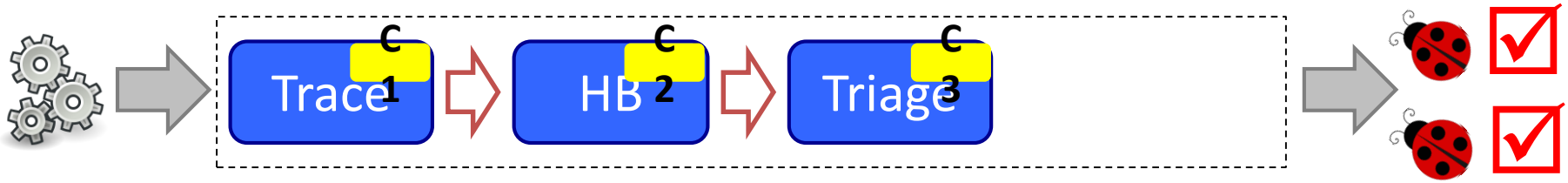
C1: How to handle the huge amount of mem accesses?

C2: What's the happens-before model?

Challenges

Local concurrency bug detection





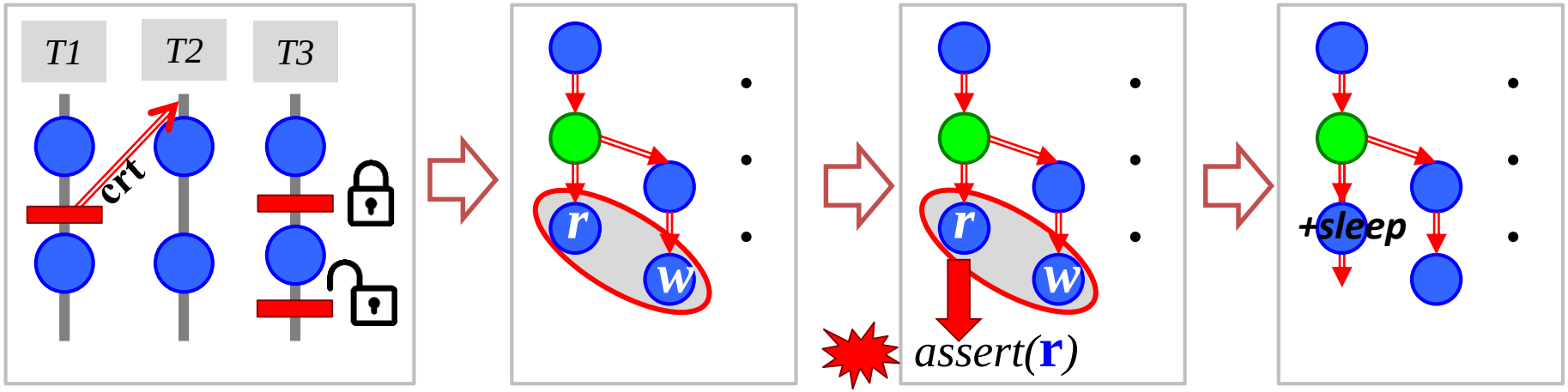
C1: How to handle the huge amount of mem accesses?

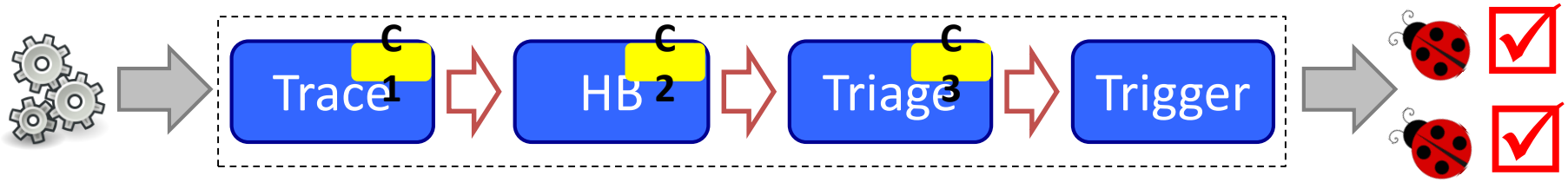
C2: What's the happens-before model?

Challenges

C3: How to estimate the distributed impact of a race?

Local concurrency bug detection





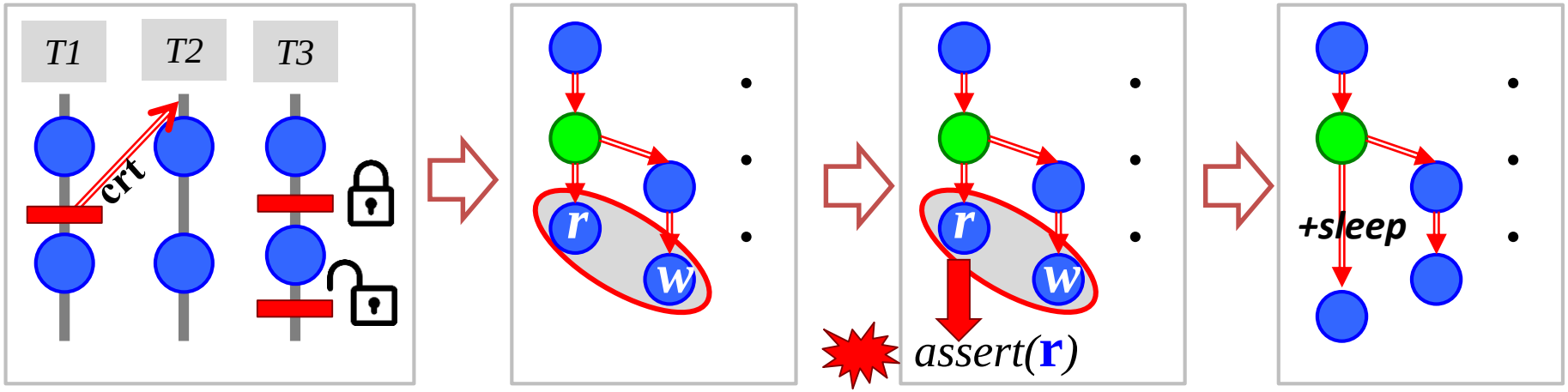
C1: How to handle the huge amount of mem accesses?

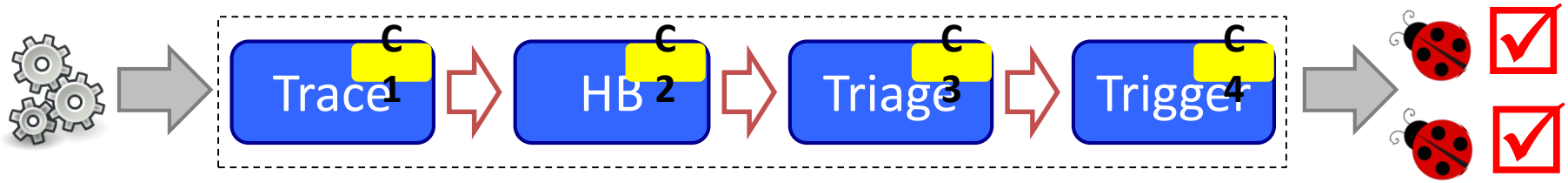
C2: What's the happens-before model?

Challenges

C3: How to estimate the distributed impact of a race?

Local concurrency bug detection





C1: How to handle the huge amount of mem accesses?

C2: What's the happens-before model?

Challenges

C3: How to estimate the distributed impact of a race?

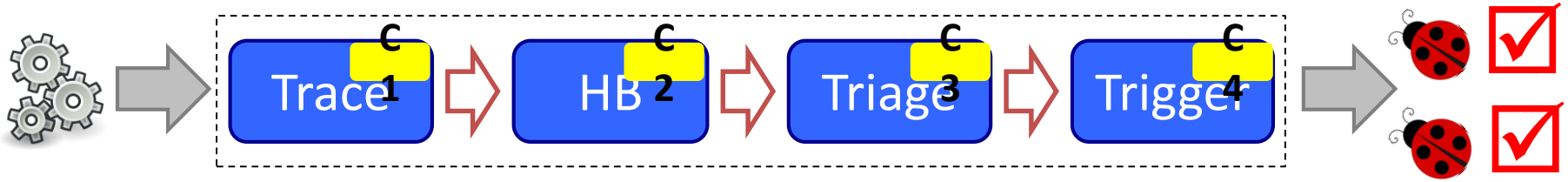
C4: How to trigger with distributed time manipulation?

Contribution

- A **comprehensive HB Model** for distributed systems

Contribution

- A **comprehensive HB Model** for distributed systems
- DCatch tool detects DCbugs from **correct runs**



C1: How to handle the huge amount of mem accesses?

C2: What's the happens-before model?

Challenges
Solved by
DCatch

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?

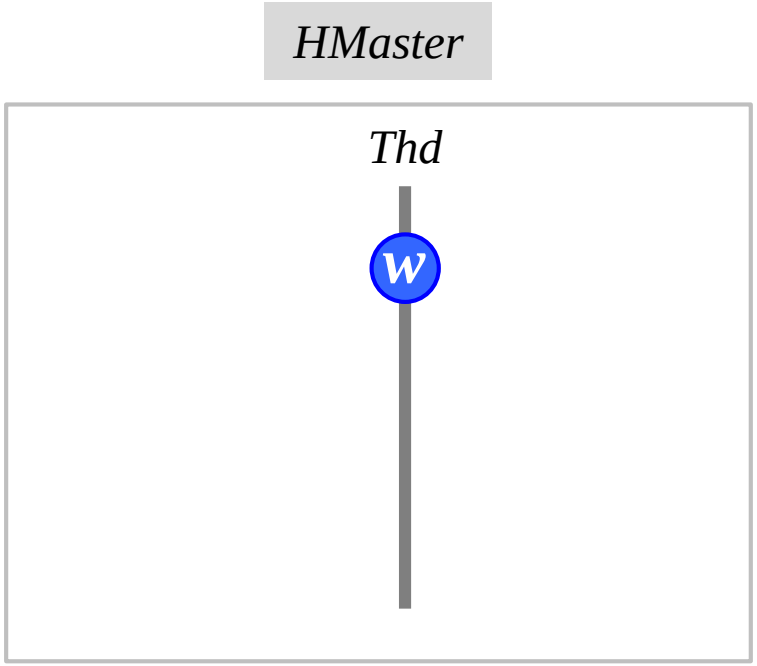
Contribution

- A **comprehensive HB Model** for distributed systems
- DCatch tool detects DCbugs from **correct runs**
- Evaluate on **4** systems
- Report **32** DCbugs, with **20** of them being truly harmful

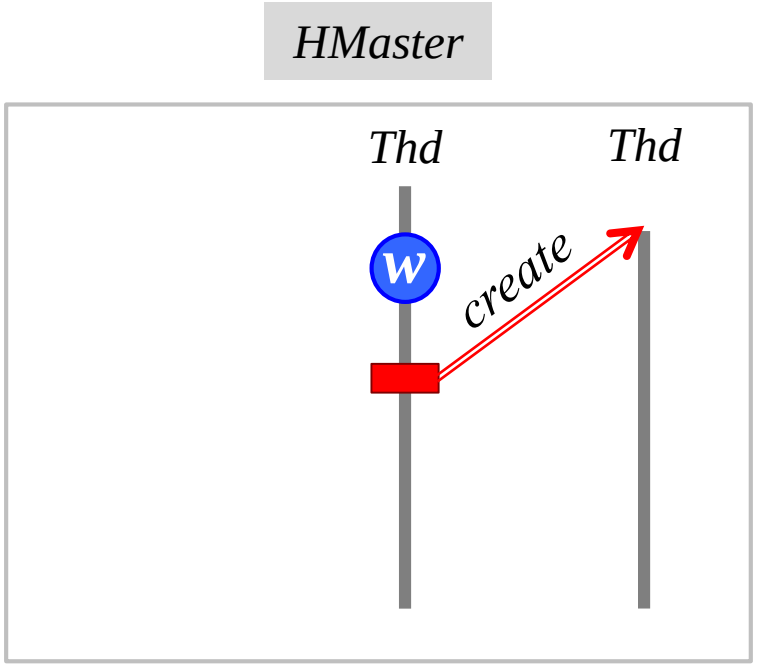
Outline

- Motivation
- DCatch Happens-before Model
- DCatch tool
- Evaluation
- Conclusion

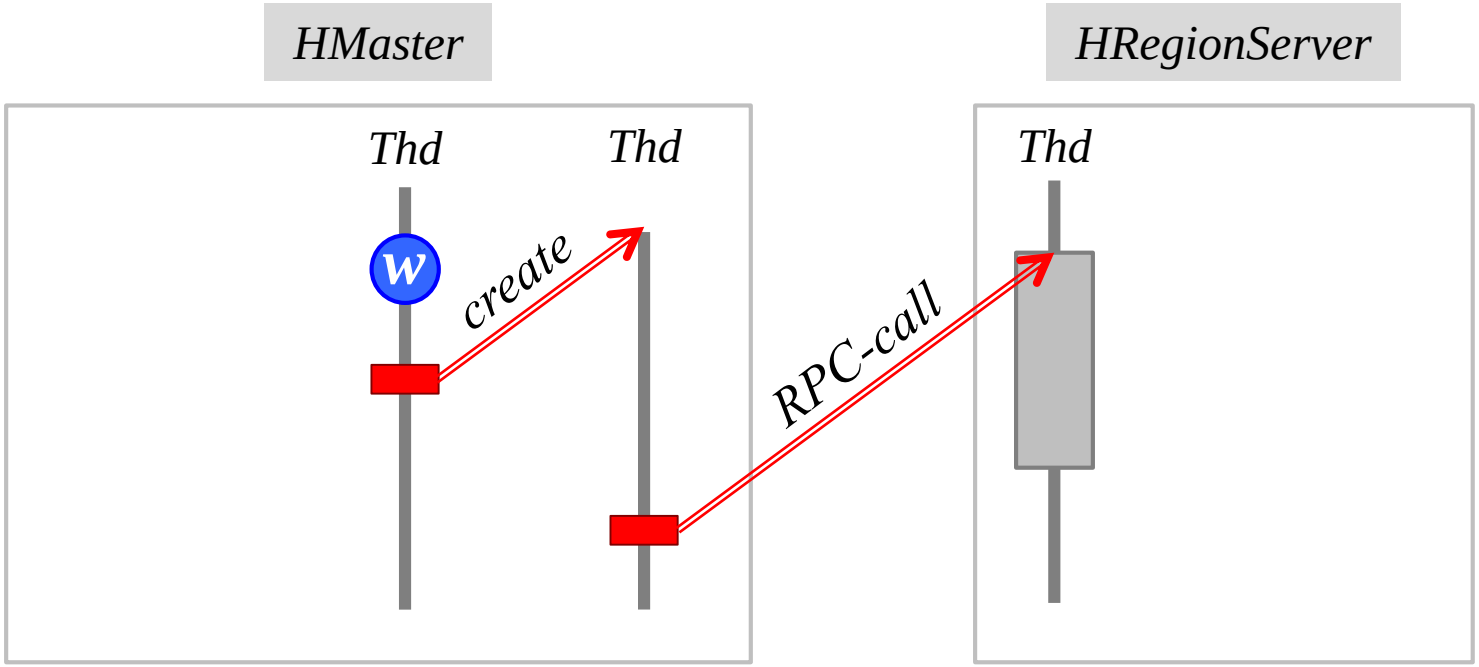
DCatch Happens-before Model



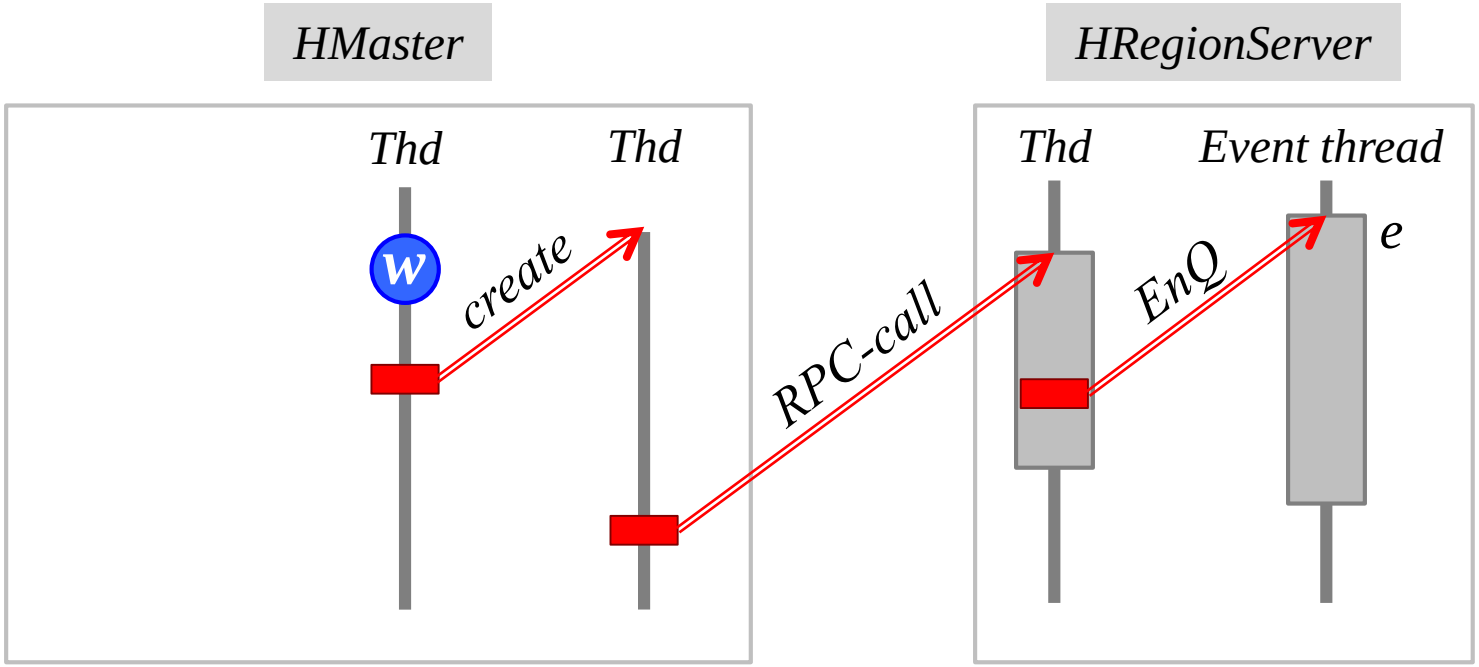
DCatch Happens-before Model



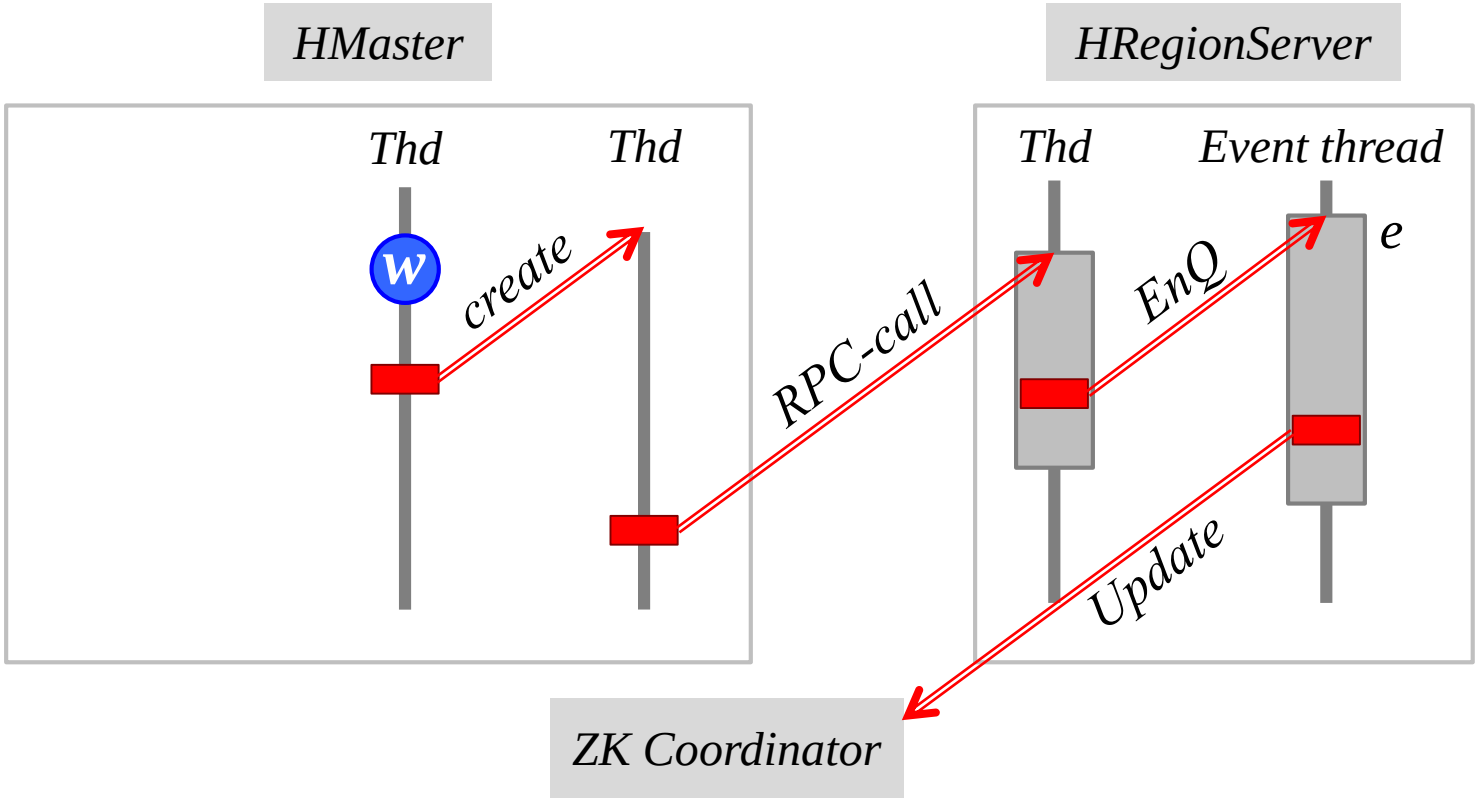
DCatch Happens-before Model



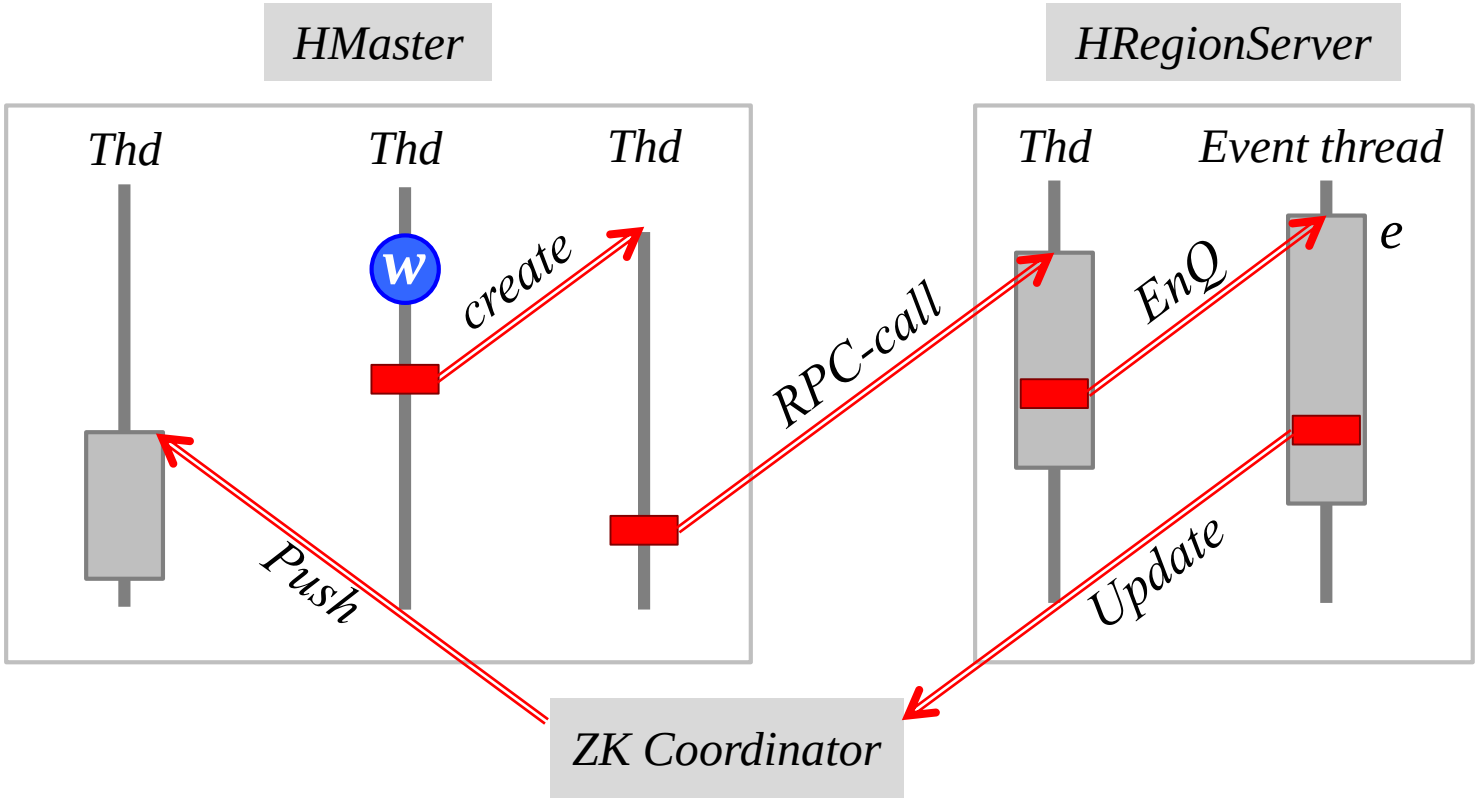
DCatch Happens-before Model



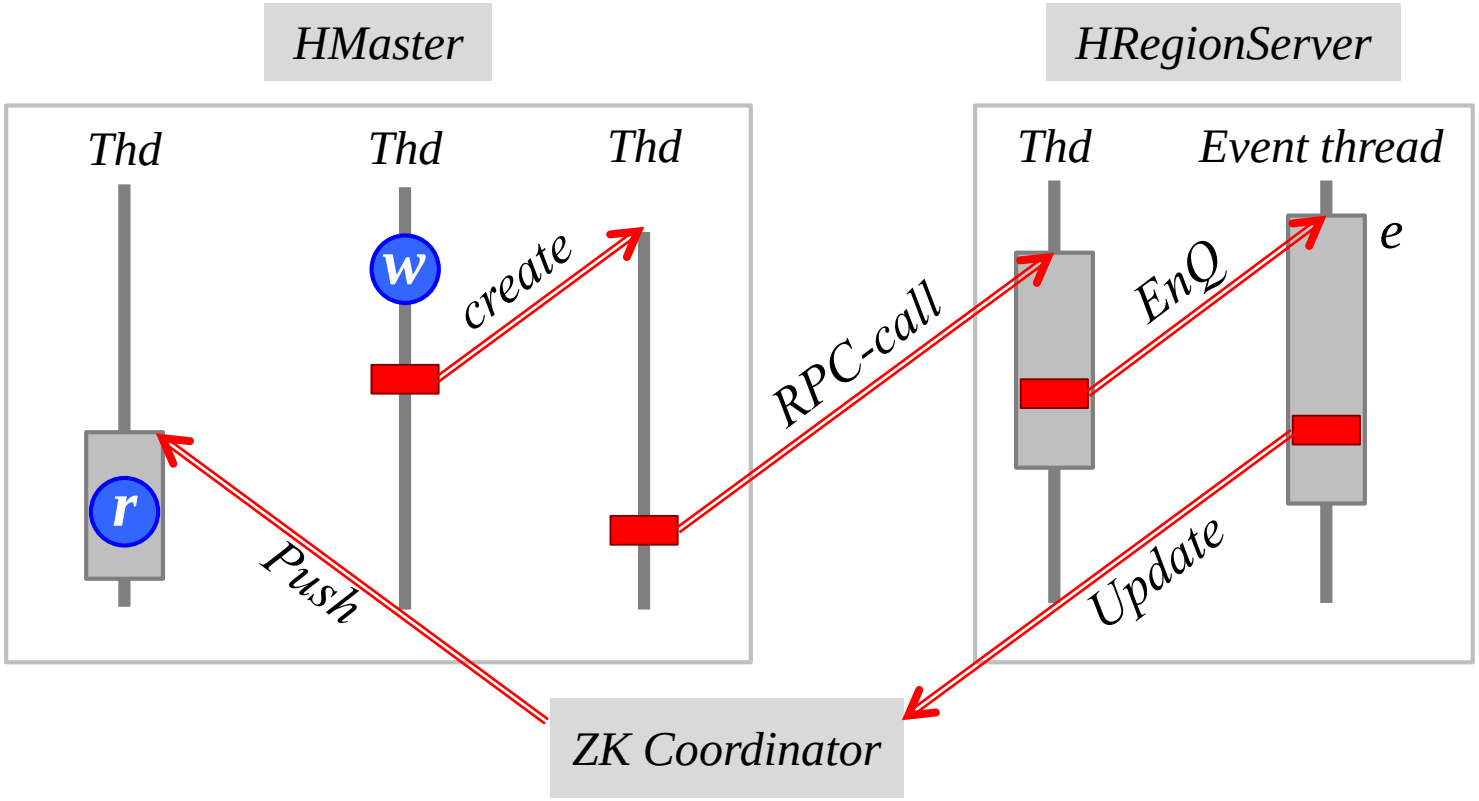
DCatch Happens-before Model



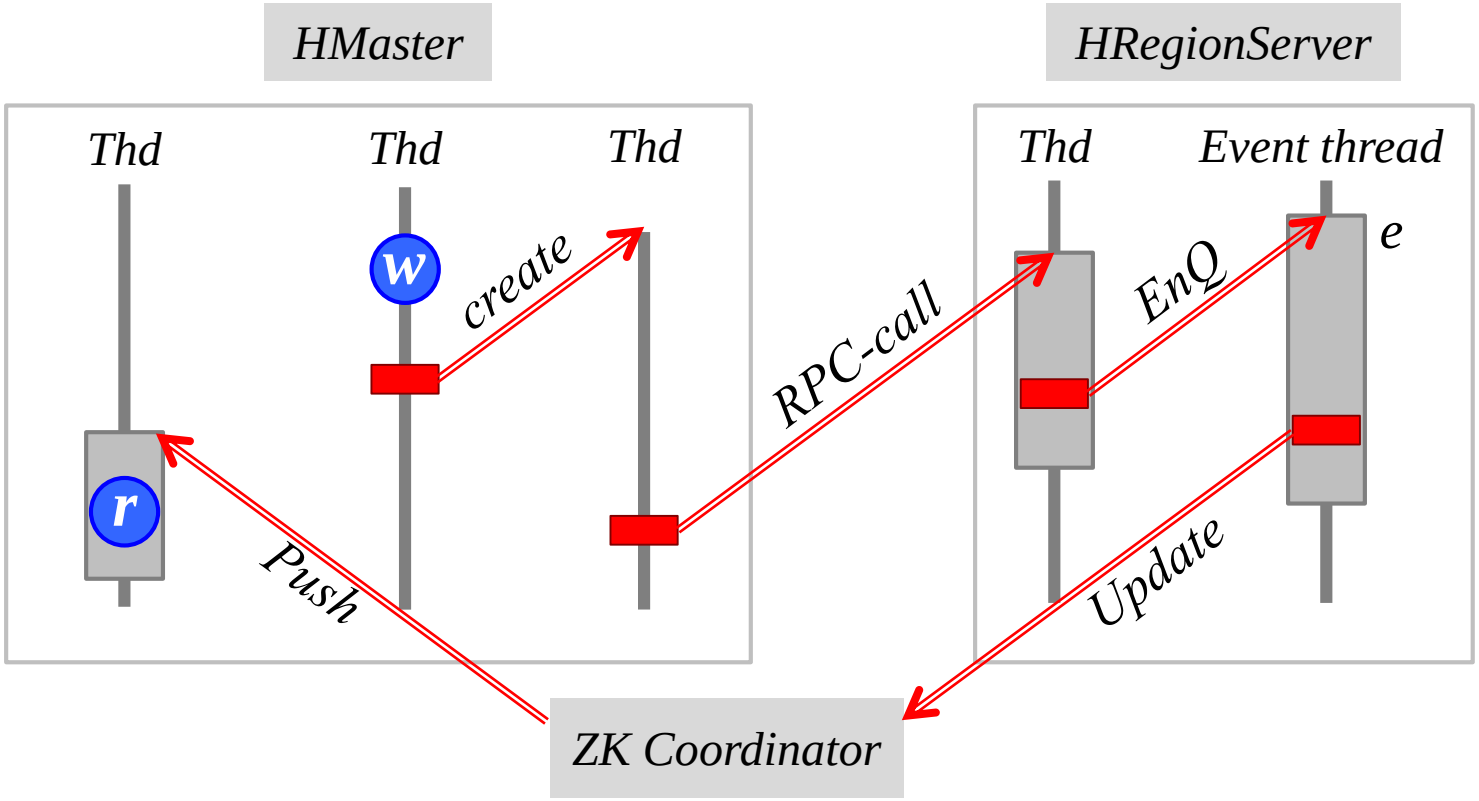
DCatch Happens-before Model



DCatch Happens-before Model

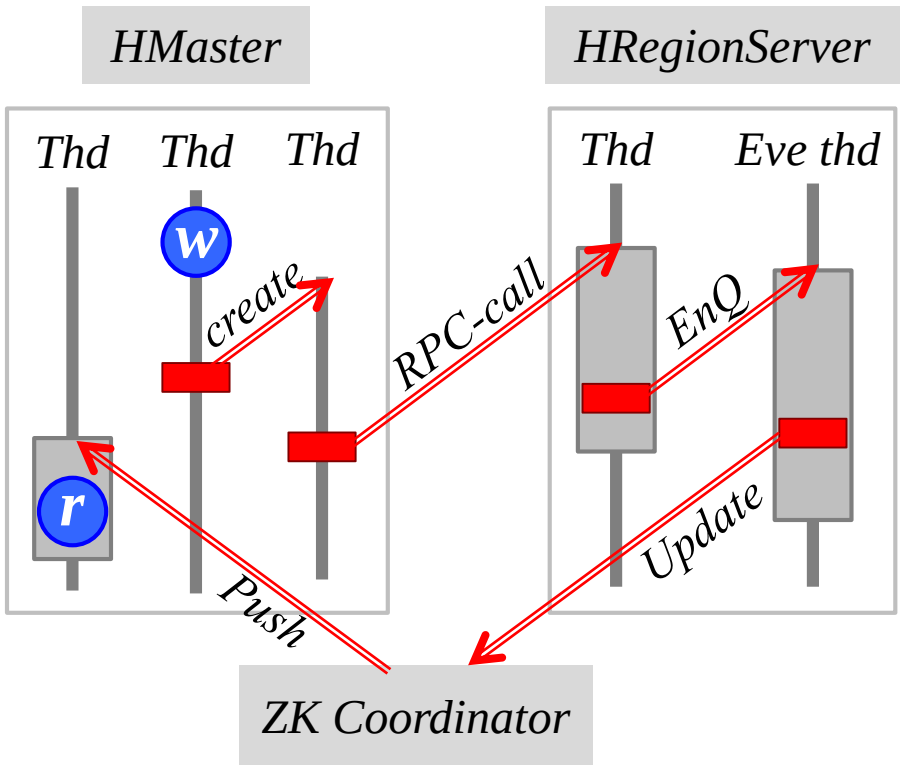


DCatch Happens-before Model

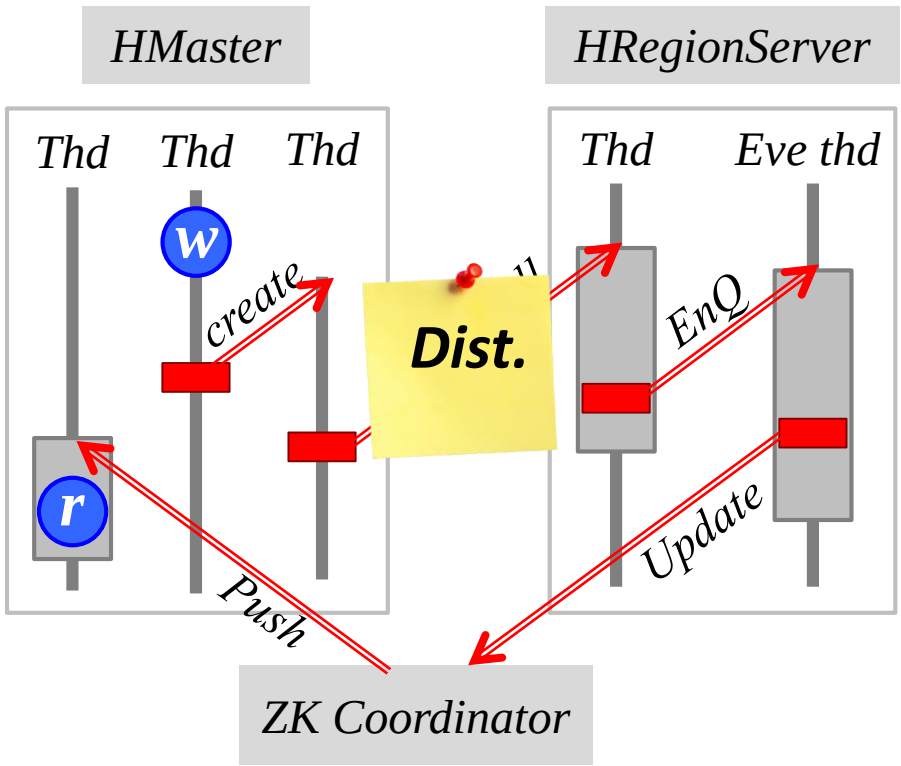


Where is HB model for distributed systems?

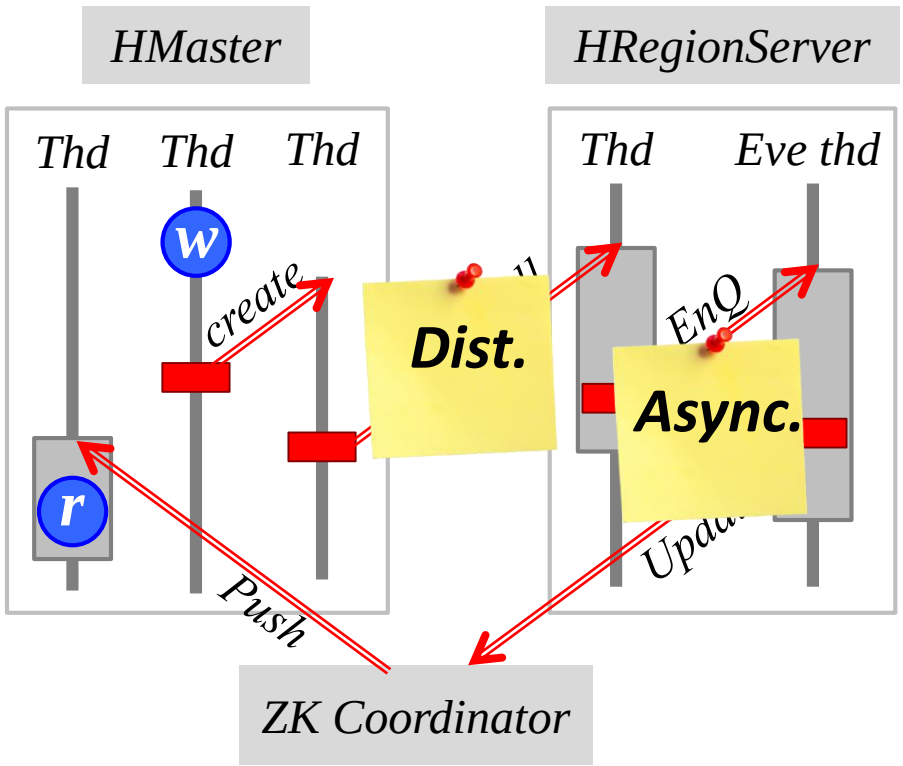
DCatch Happens-before Model

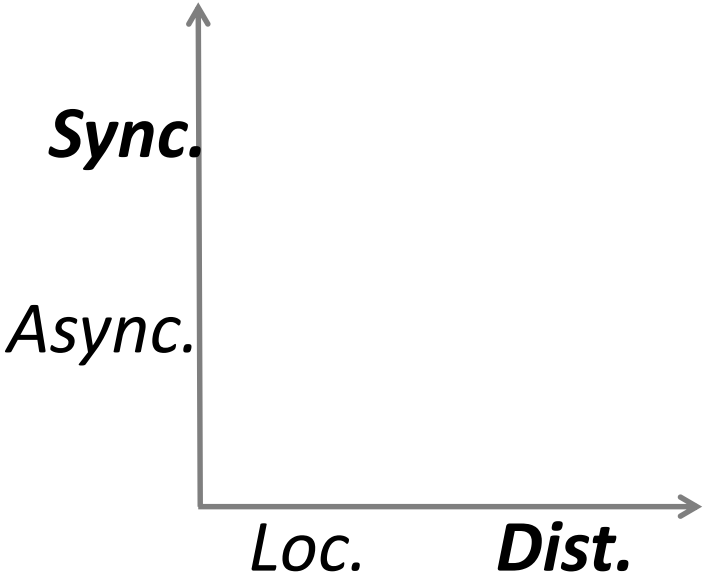


DCatch Happens-before Model

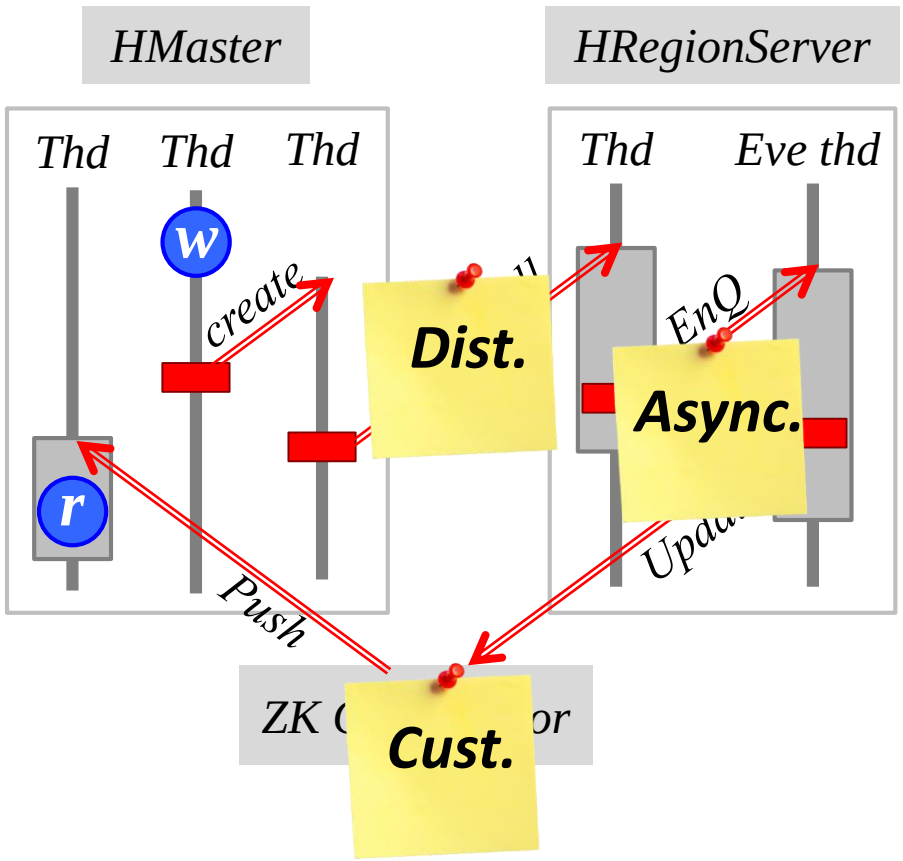


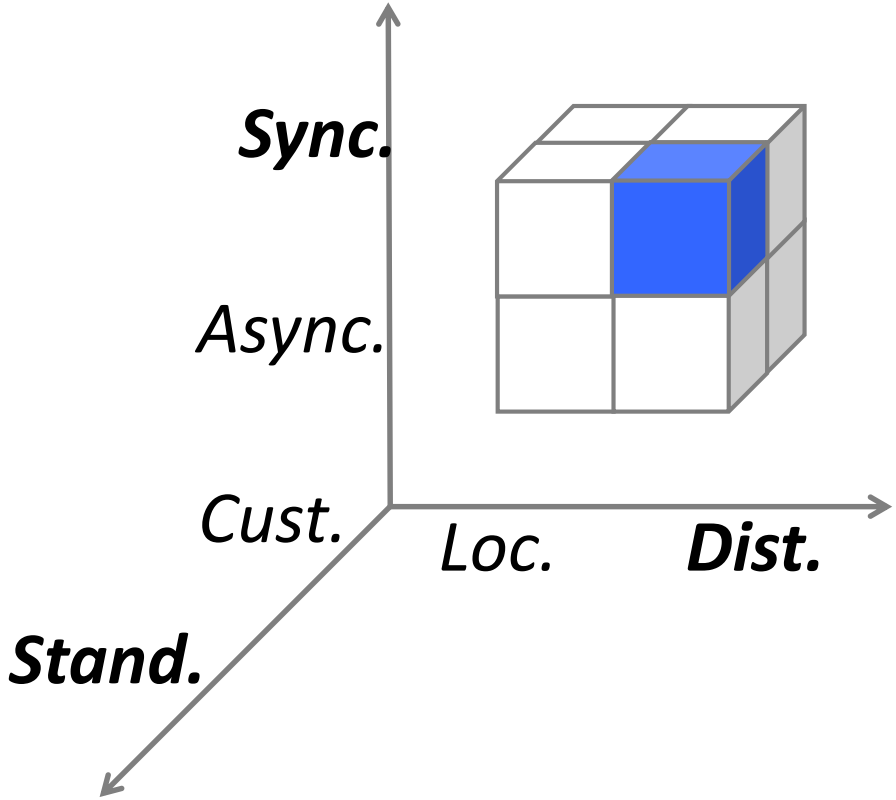
DCatch Happens-before Model



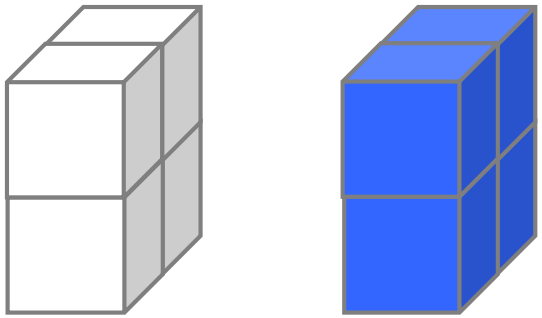


DCatch Happens-before Model





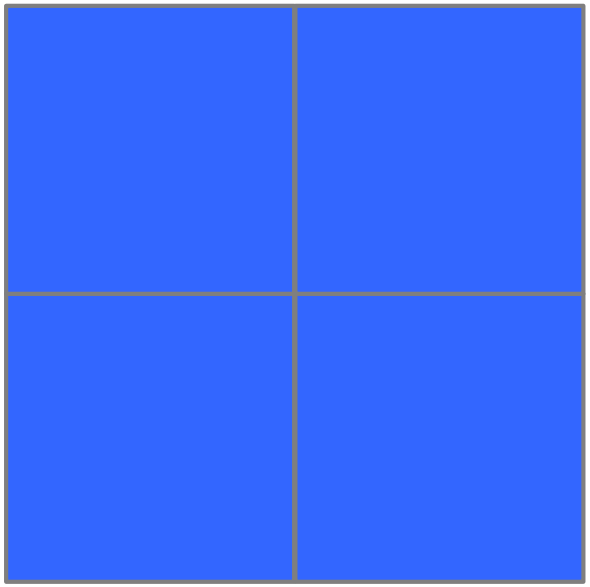
Distributed rules



Local ***Distributed***

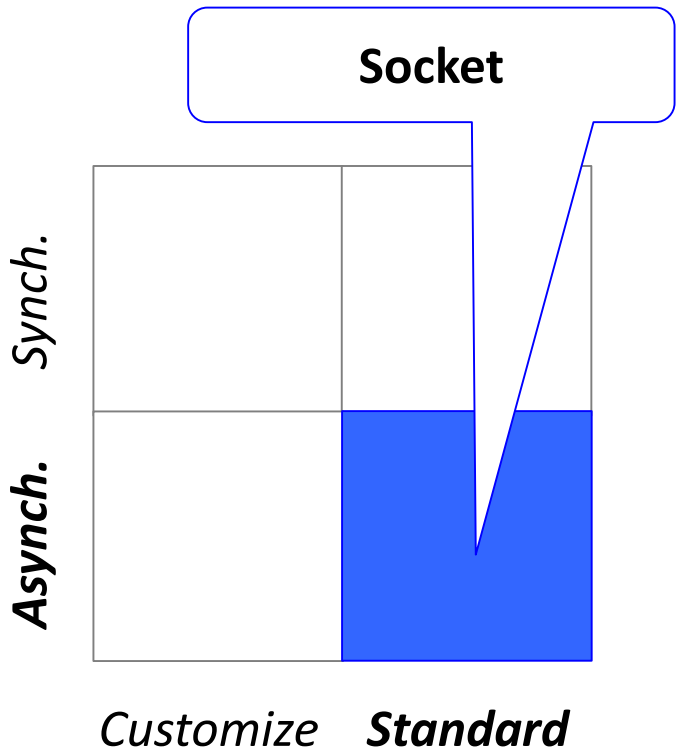
Sync.

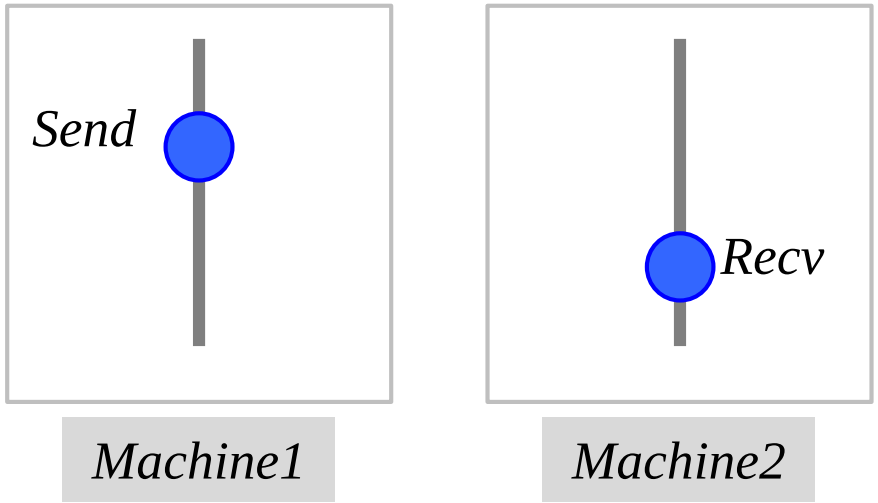
Async.



Custom *Standard*

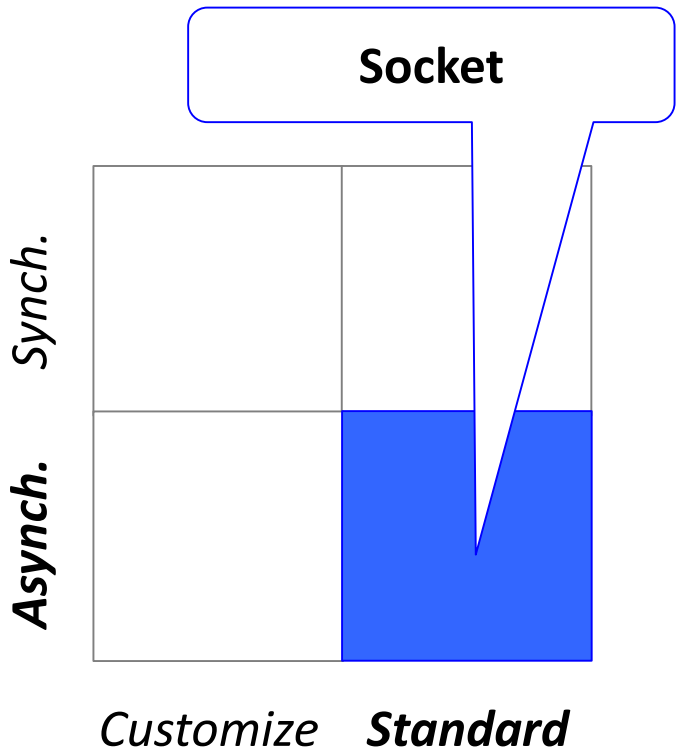
Distributed rule #1

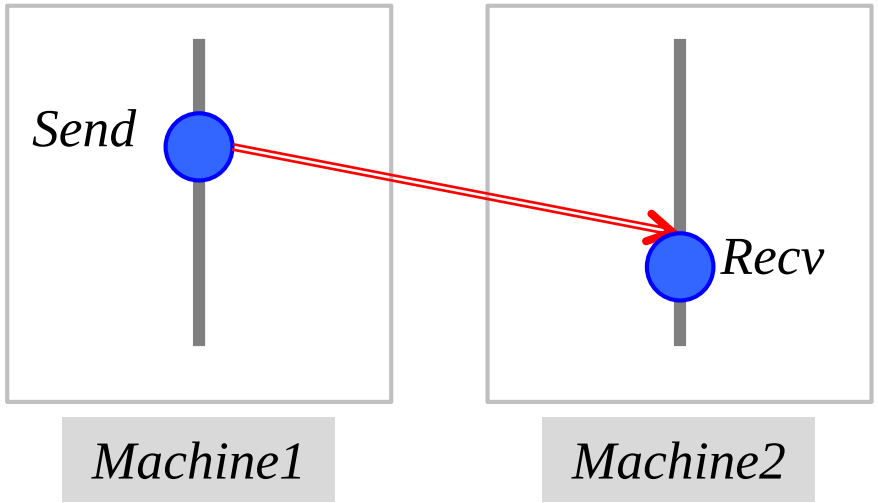




- Logical time clock (Leslie Lamport, 1978)

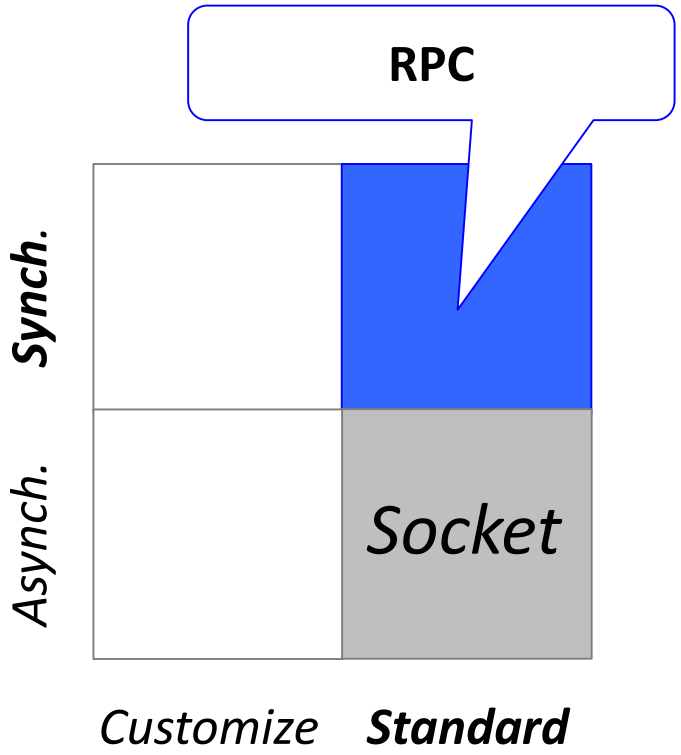
Distributed rule #1

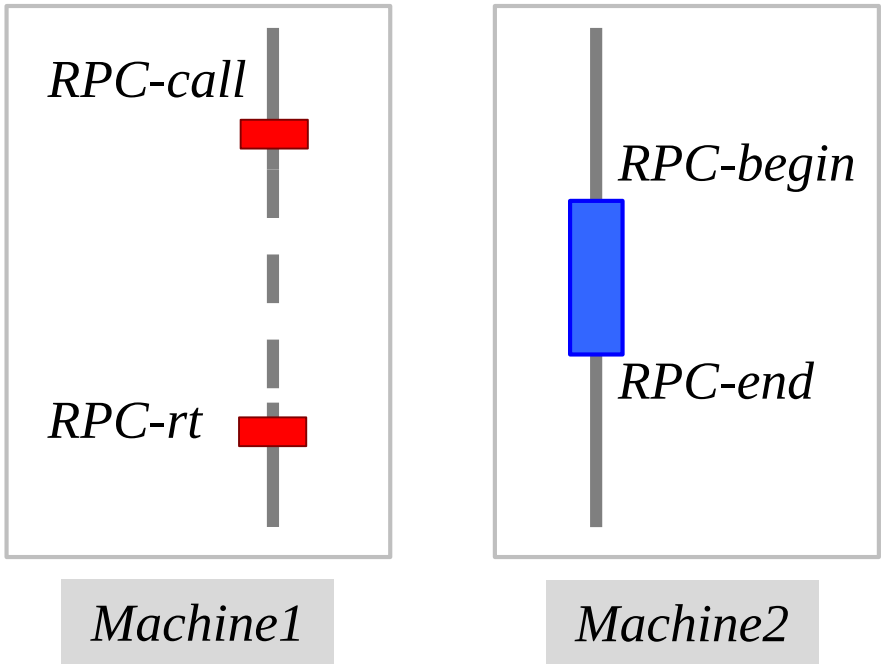




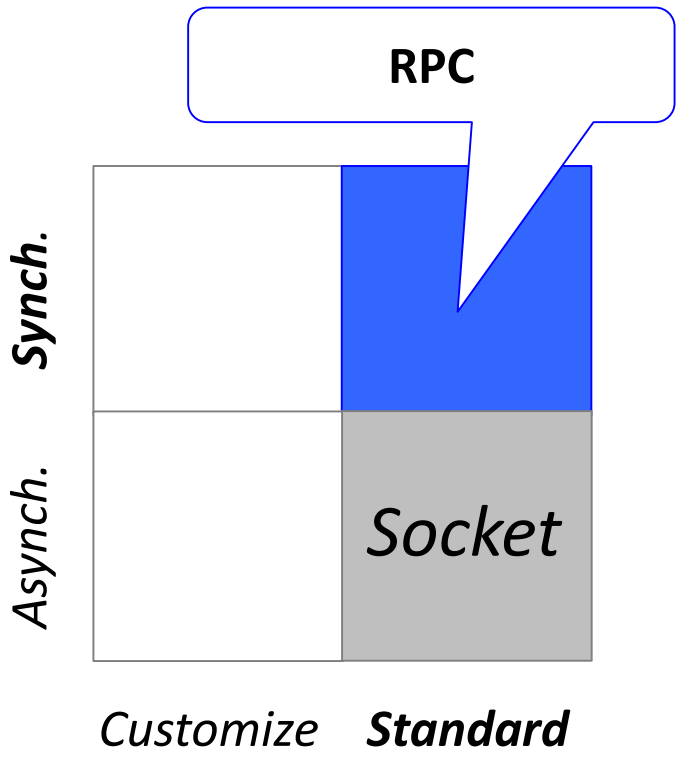
- Logical time clock (Leslie Lamport, 1978)

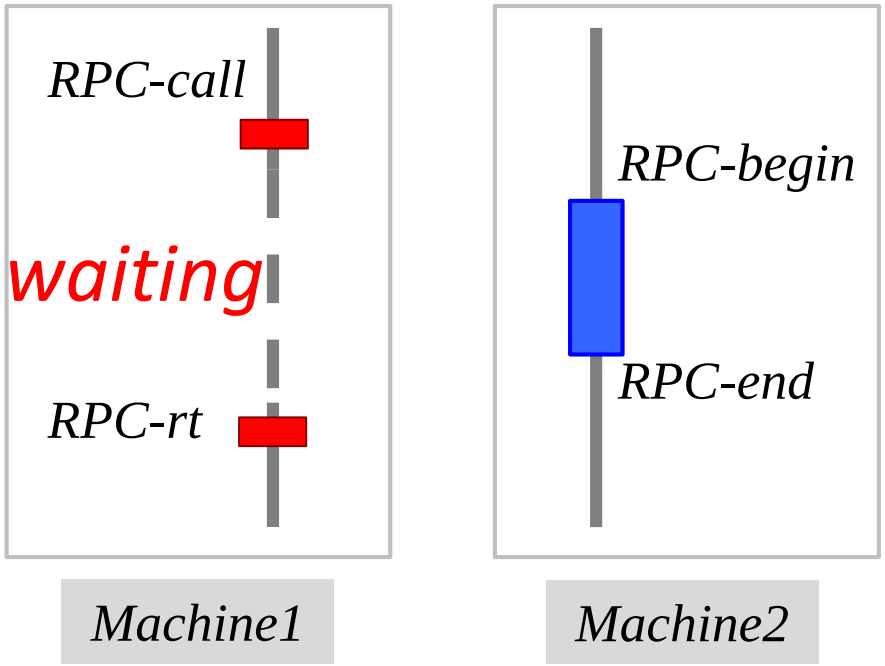
Distributed rule #2



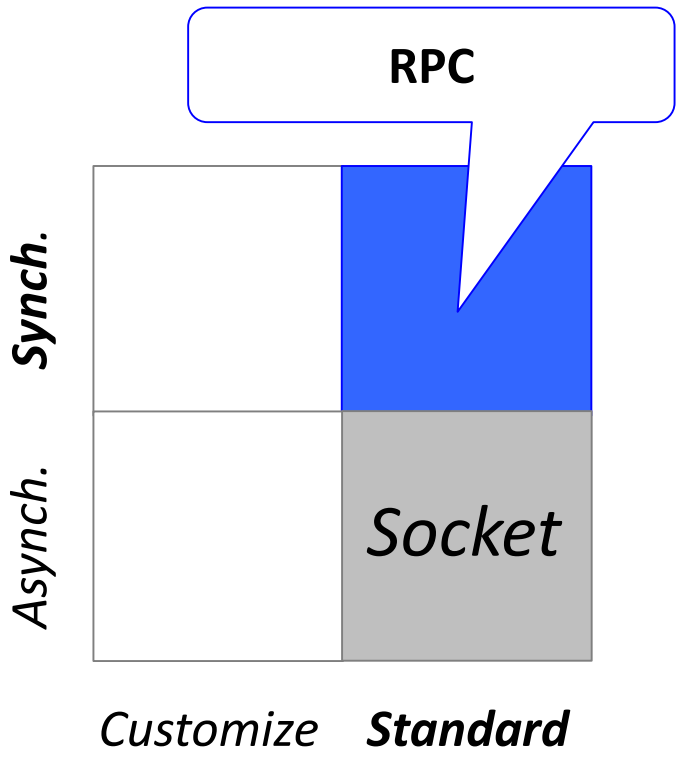


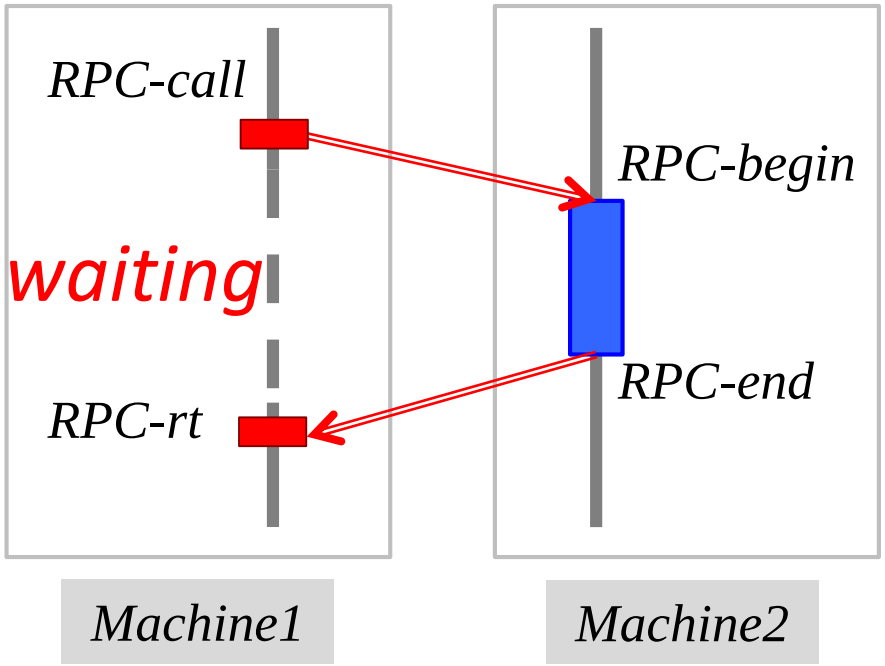
Distributed rule #2



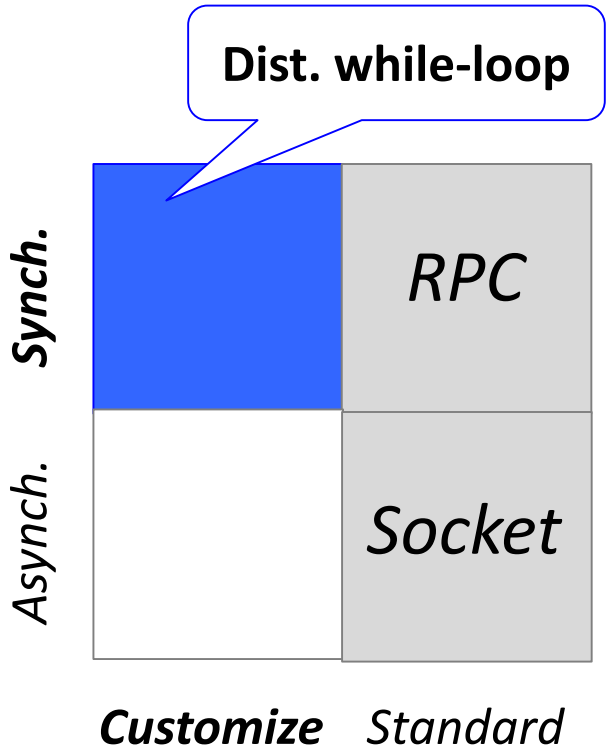


Distributed rule #2





Distributed rule #3

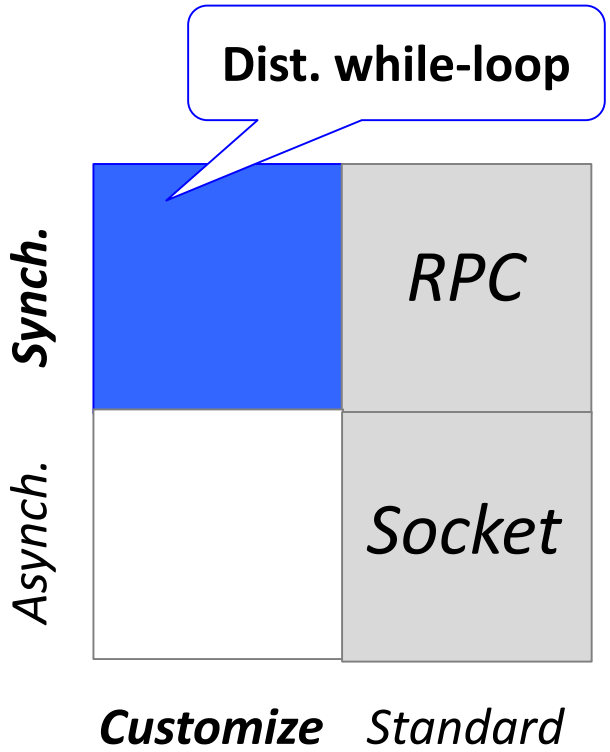


In multi-threaded systems:

```
//Thread1
flag = True;
```

```
//Thread
while(!flag){
}
...
```

Distributed rule #3

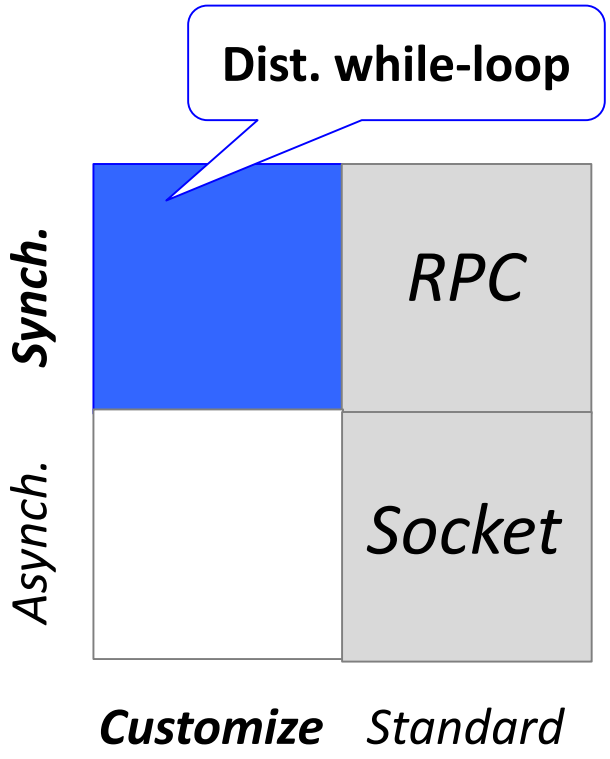


In multi-threaded systems:

```
//Thread1
flag = True;
```

```
//Thread
while(!flag){
}
...
```

Distributed rule #3



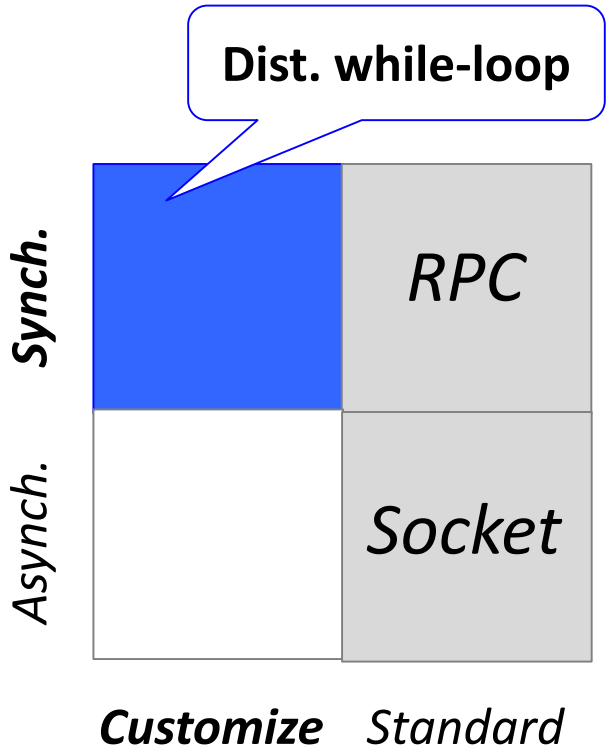
In multi-threaded systems:

```
//Thread1
flag = True;
```

```
//Thread
while(!flag){
}
...
```

In distributed systems:

Distributed rule #3



In distributed systems:

Machine A

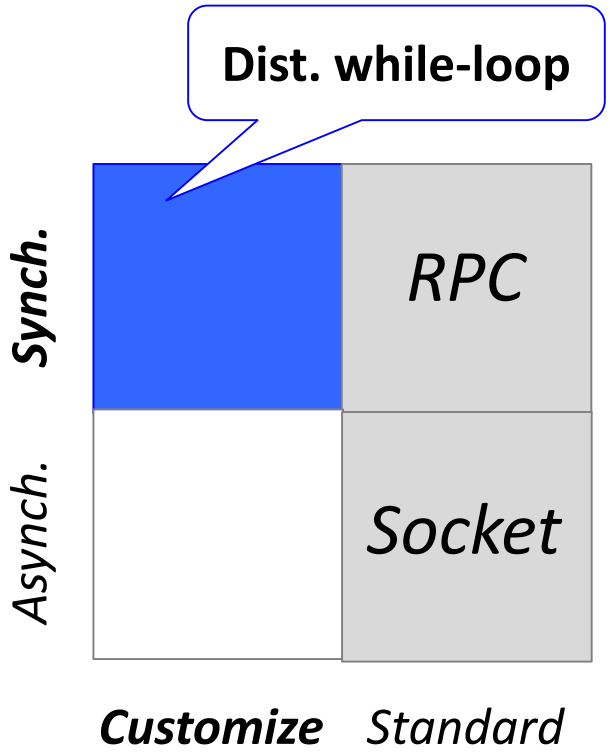
```
//Thread1
flag = True;
```

```
//Thread2
bool getFlag(){
   return flag;
}
```

Machine B

```
//Thread
while(!getFlag()){
}
...
```

Distributed rule #3



In distributed systems:

Machine A

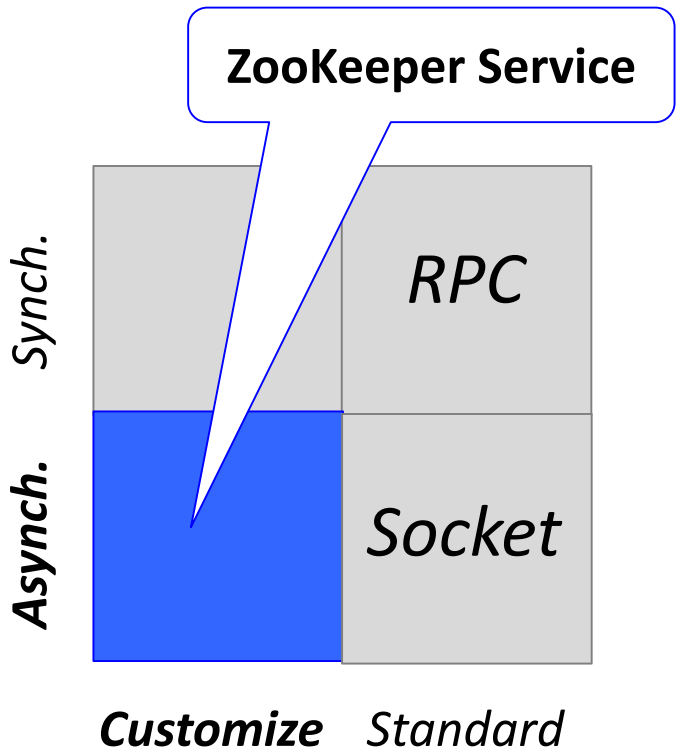
```
//Thread1
flag = True;
```

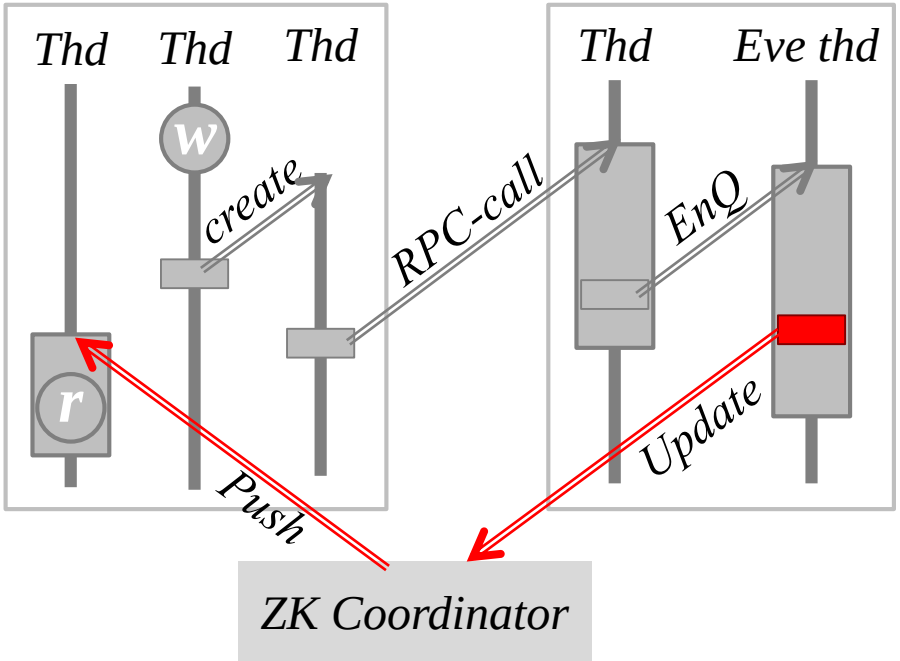
```
//Thread2
bool getFlag(){
   return flag;
}
```

Machine B

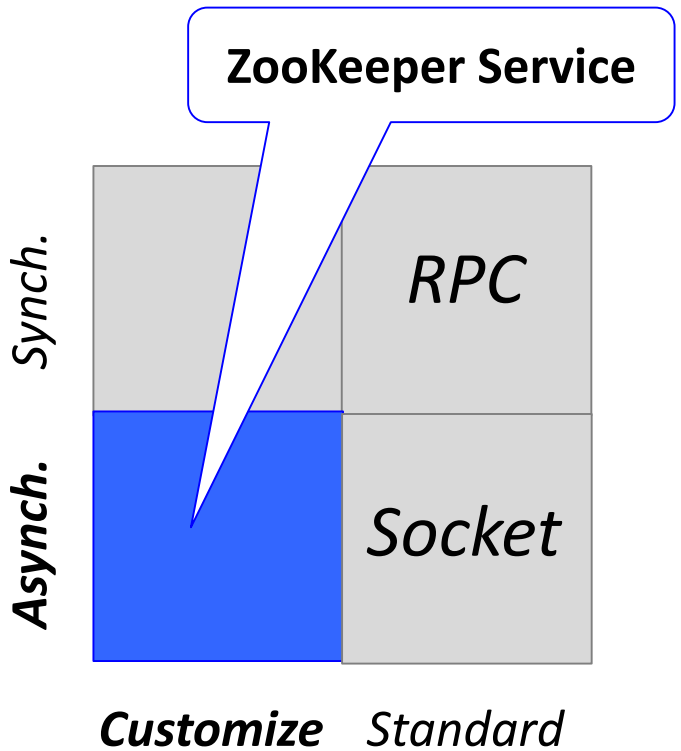
```
//Thread
while(!getFlag()){
}
...
```

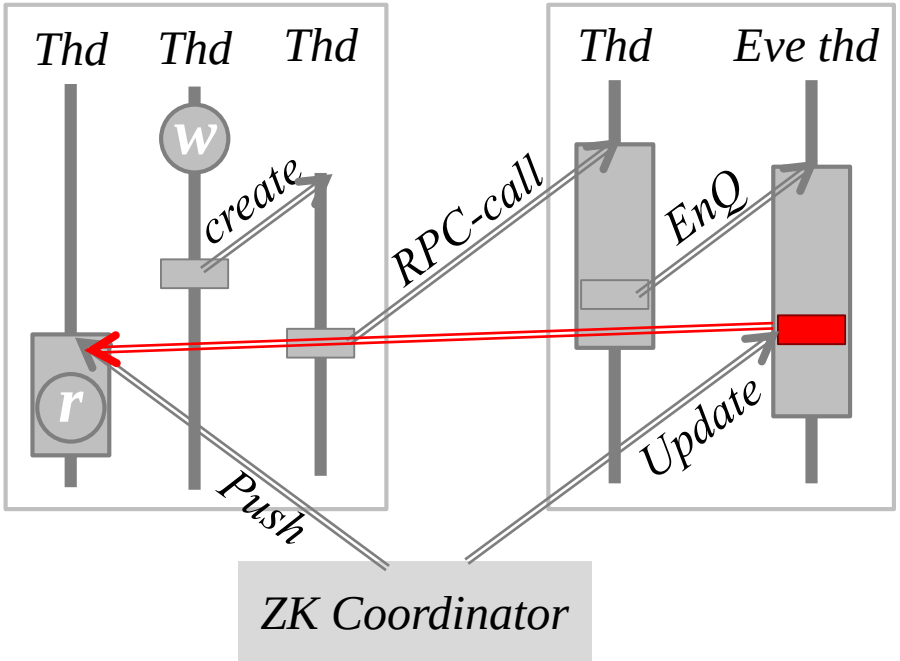
Distributed rule #4





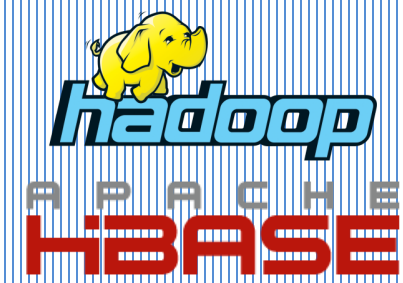
Distributed rule #4

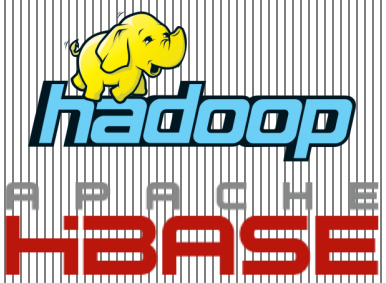


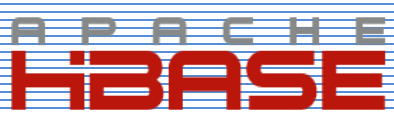


Distributed rules

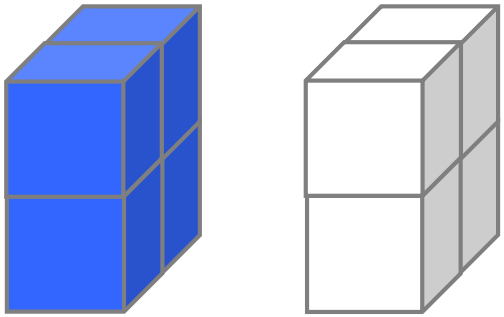
	Customized	Standard
Synchronous	*Dist. While-loop*	*RPC*
Asynchronous	*Zookeeper Service*	*Socket*





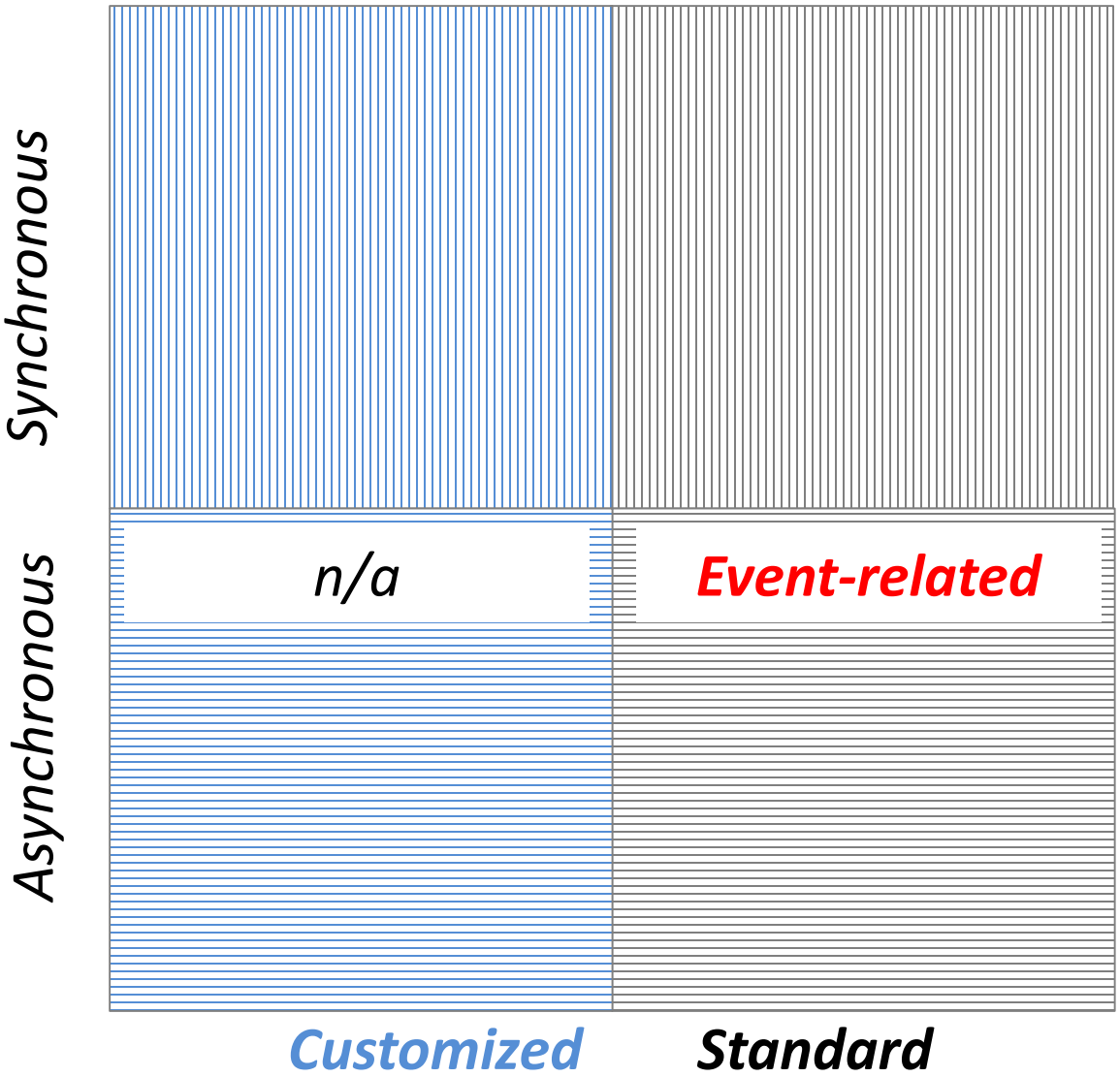


Local rules

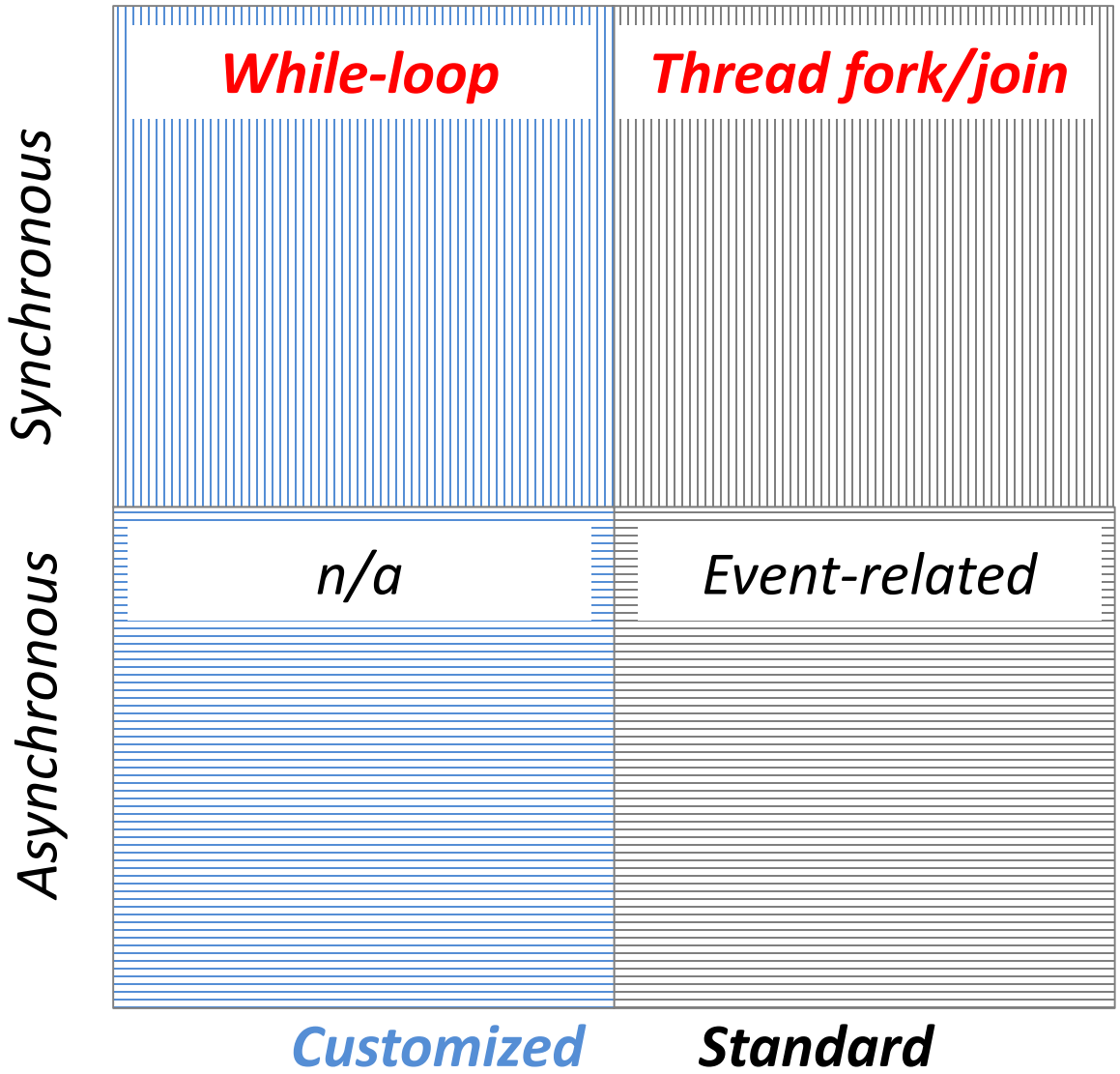


Local *Distributed*

Local rules


Synchronous
Asynchronous
n/a
Event-related
Customized
Standard

Local rules


While-loop
Thread fork/join
Synchronous
n/a
Event-related
Asynchronous
Customized
Standard

Local rules

	Customized	Standard
Synchronous	*While-loop*	*Thread fork/join*
Asynchronous	*n/a*	*Event-related*

Outline

- Motivation
- DCatch Happens-before Model
- DCatch tool
- Evaluation
- Conclusion

Trace → HB → Triage → Trigger

Trace → HB → Triage → Trigger

C1: How to handle the huge amount of mem accesses?

C2: What's the happens-before model?

Challenges

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?

Selective tracing: only mem accesses in Event/message handlers and their callee.

Trace → **HB** → Triage → Trigger

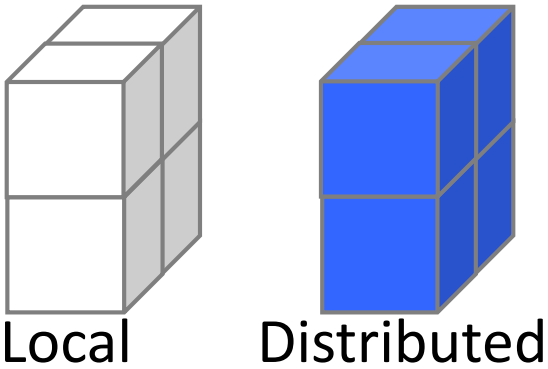
C1: How to handle the huge amount of mem accesses?

C2: What's the happens-before model?

Challenges

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?



[1] Raychev. Effective Race Detection for Event-Driven Programs. In OOPSLA'13

Trace → HB → Triage → Trigger

Challenges

C1: How to handle the huge amount of mem accesses?

C2: What's the happens-before model?

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?

Machine B

```
//RPC thread
Task getTask(jID){
  ...
  return jMap.get(jID);
}
```

```
//UnReg thread
void unReg(jID){
  jMap.remove(jID);
  ....
}
```

Trace → HB → **Triage** → Trigger

Challenges

C1: How to handle the huge amount of mem accesses?

C2: What's the happens-before model?

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?

Machine A

```
while(!getTask(jID)){
}
```

Machine B

```
//RPC thread
Task getTask(jID){
  ...
  return jMap.get(jID);
}
```

```
//UnReg thread
void unReg(jID){
  jMap.remove(jID);
  ....
}
```

Trace → HB → Triage → Trigger

Challenges

C1: How to handle the huge amount of mem accesses?

C2: What's the happens-before model?

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?

Machine A

```
while(!getTask(jID)){
}
```

Machine B

```
//RPC thread
Task getTask(jID){
  ...
  return jMap.get(jID);
}
```

```
//UnReg thread
void unReg(jID){
  jMap.remove(jID);
  ....
}
```

Trace → HB → Triage → Trigger

Challenges

C1: How to handle the huge amount of mem accesses?

C2: What's the happens-before model?

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?

Machine A

```
while(!getTask(jID)){
}
```

Machine B

```
//RPC thread
Task getTask(jID){
  ...
  return jMap.get(jID);
}
```

```
//UnReg thread
void unReg(jID){
  jMap.remove(jID);
  ....
}
```

Trace → HB → Triage → **Trigger**

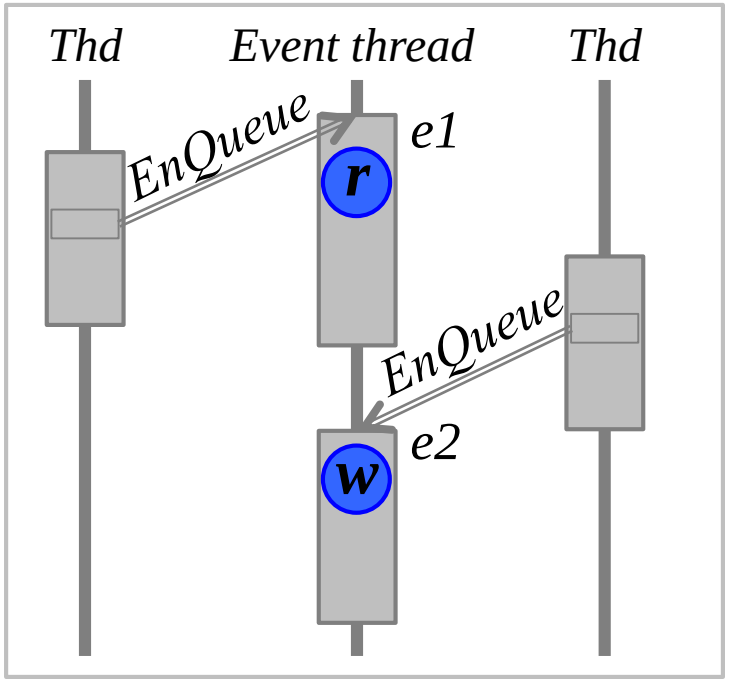
C1: How to handle the huge amount of mem accesses?

C2: What's the happens-before model?

Challenges

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?



Trace → HB → Triage → **Trigger**

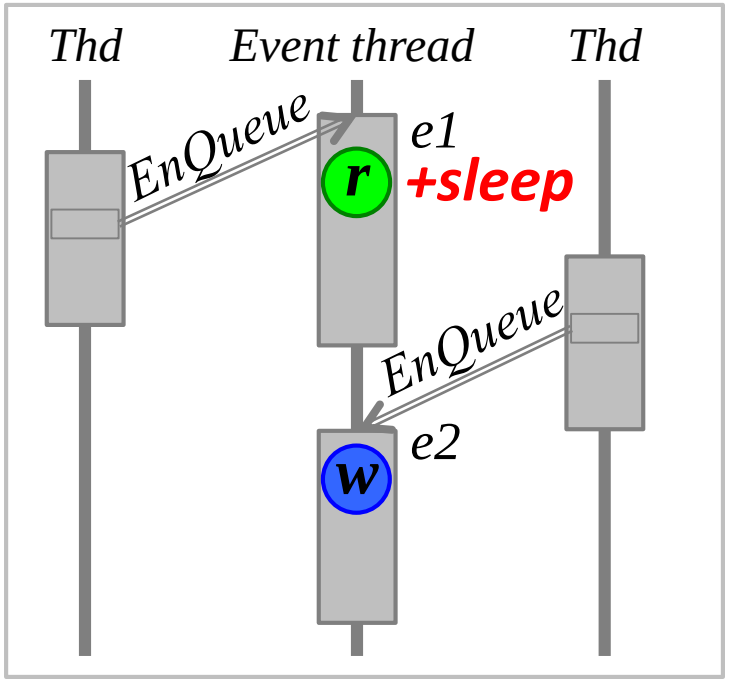
C1: How to handle the huge amount of mem accesses?

C2: What's the happens-before model?

Challenges

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?



Trace → HB → Triage → **Trigger**

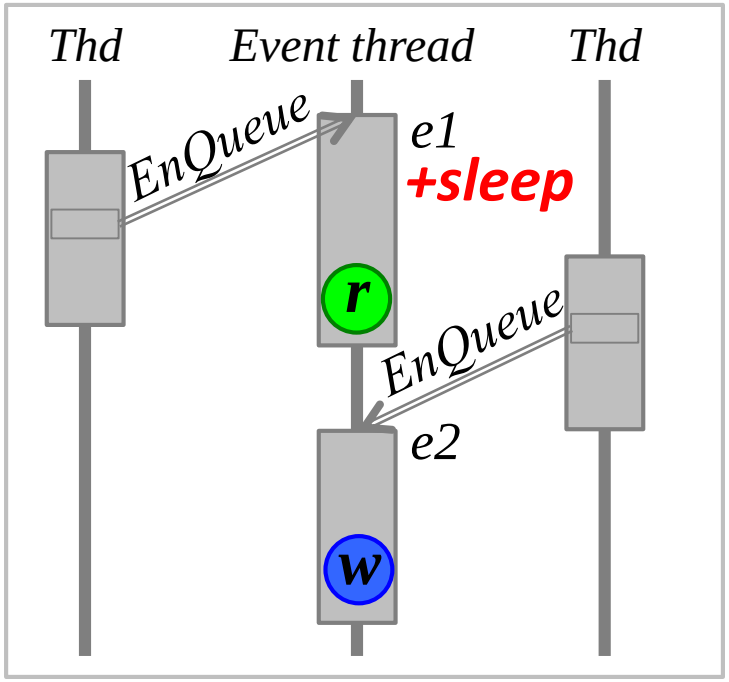
C1: How to handle the huge amount of mem accesses?

C2: What's the happens-before model?

Challenges

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?



Trace → HB → Triage → **Trigger**

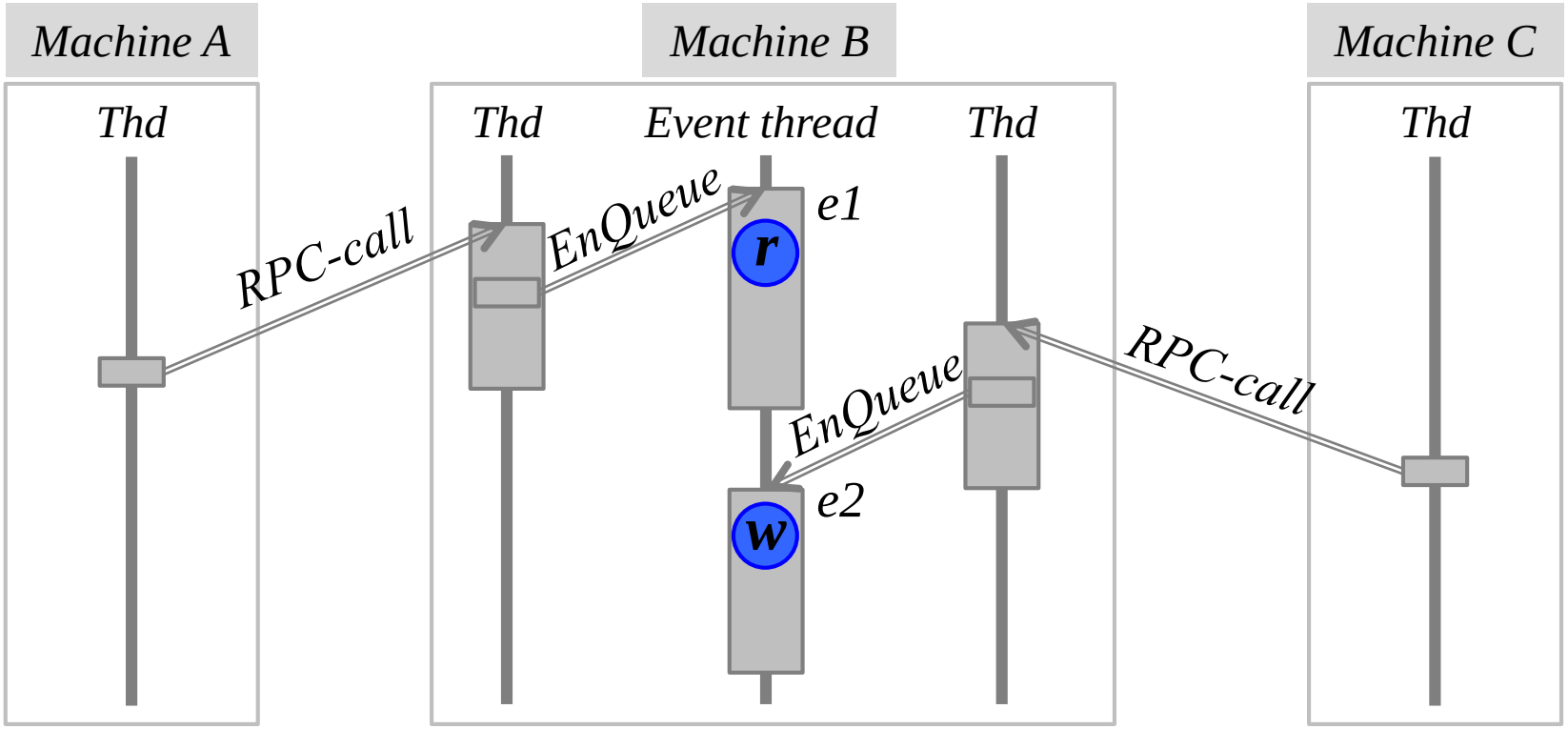
C1: How to handle the huge amount of mem accesses?

C2: What's the happens-before model?

Challenges

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?



Trace → HB → Triage → **Trigger**

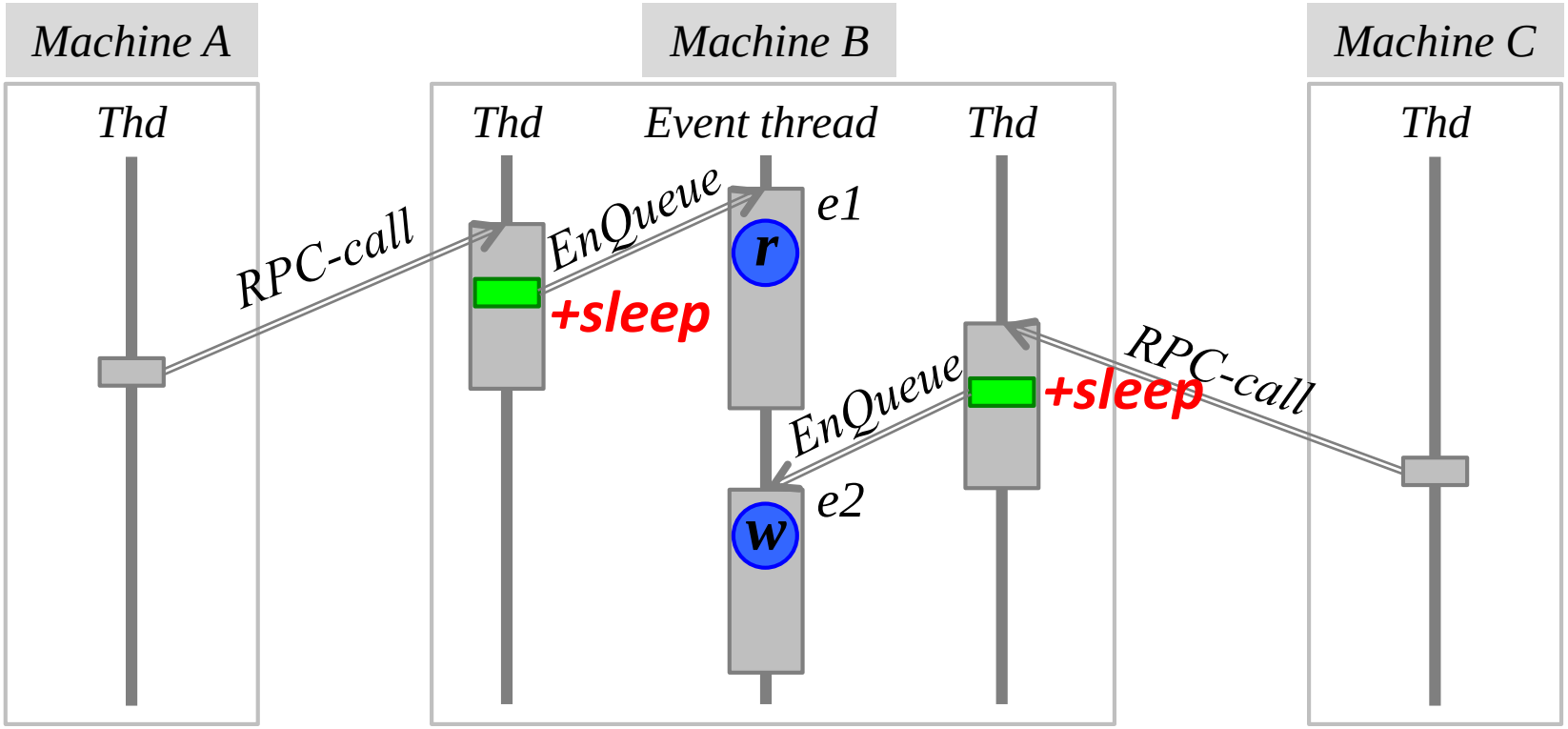
C1: How to handle the huge amount of mem accesses?

C2: What's the happens-before model?

Challenges

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?



Trace → HB → Triage → **Trigger**

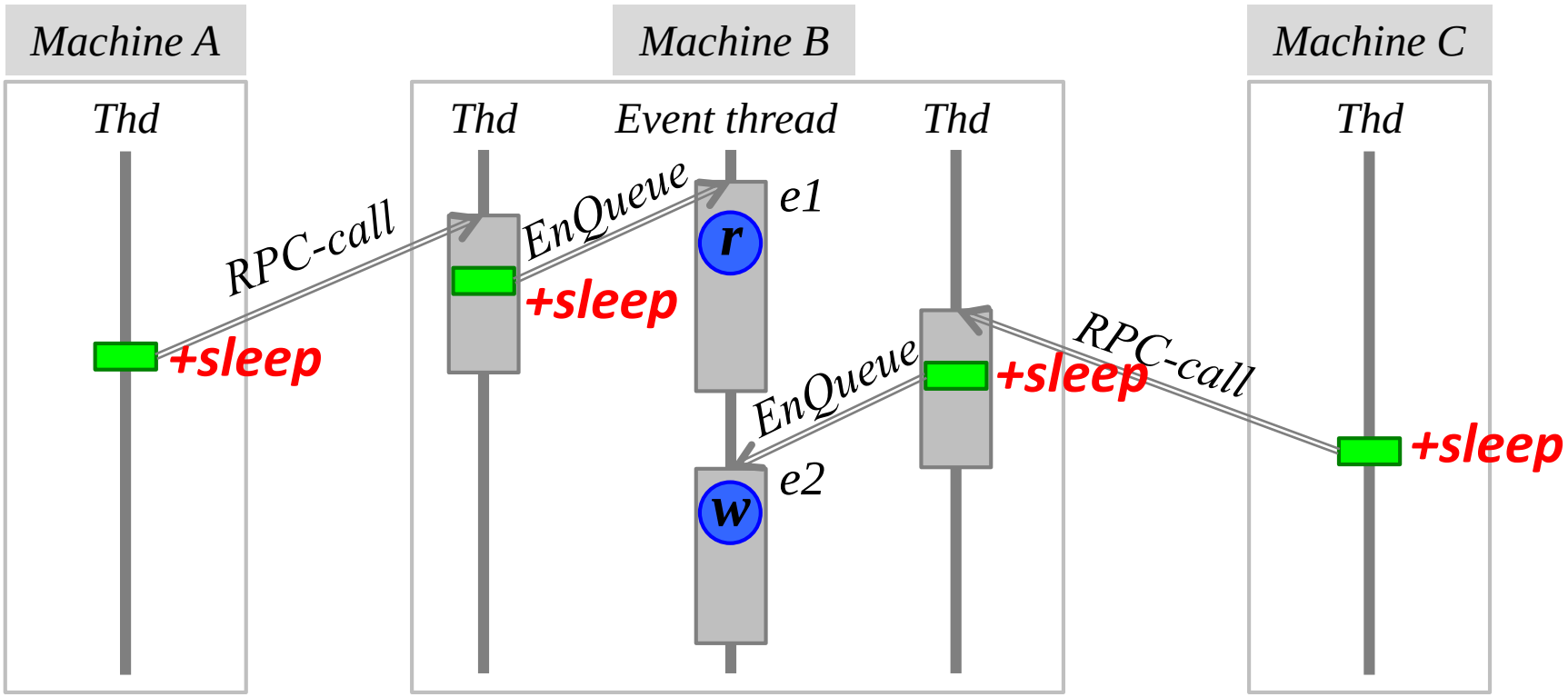
C1: How to handle the huge amount of mem accesses?

C2: What's the happens-before model?

Challenges

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?



Outline

- Motivation
- DCatch Happens-before Model
- DCatch tool
- Evaluation
- Conclusion

Methodology

- Benchmarks:
  - 7 real-world DCbugs from TaxDC[1]
  - 4 distributed systems

[1] Leesatapornwongsa. TaxDC. In ASPLOS'16

Overall results

BugID	Detected?	#. Bugs	#. Benign	#. false-pos
CA-1011	✔	3	0	0
HB-4539	✔	3	0	1
HB-4729	✔	4	1	0
MR-3274	✔	2	0	4
MR-4637	✔	1	2	4
ZK-1144	✔	5	1	1
ZK-1270	✔	6	2	0

Overall results

BugID	Detected?	#. Bugs	#. Benign	#. false-pos
CA-1011	✓	3	0	0
HB-4539	✓	3	0	1
HB-4729	✓	4	1	0
MR-3274	✓	2	0	4
MR-4637	✓	1	2	4
ZK-1144	✓	5	1	1
ZK-1270	✓	6	2	0

Overall results

BugID	Detected?	#. Bugs	#. Benign	#. false-pos
CA-1011	✓	3	0	0
HB-4539	✓	3	0	1
HB-4729	✓	4	1	0
MR-3274	✓	2	0	4
MR-4637	✓	1	2	4
ZK-1144	✓	5	1	1
ZK-1270	✓	6 = 12 + **8**	2	0

Overall results

BugID	Detected?	#. Bugs	#. Benign	#. false-pos
CA-1011	✓	3	0	0
HB-4539	✓	3	0	1
HB-4729	✓	4	1	0
MR-3274	✓	2	0	4
MR-4637	✓	1	2	4
ZK-1144	✓	5	1	1
ZK-1270	✓	6	2	0

Other results in our paper

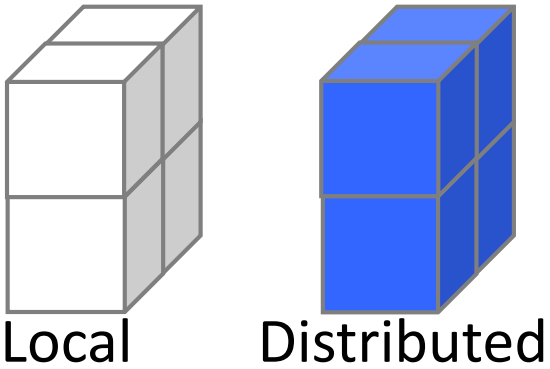
- Performance overhead
- Trace compositions
- HB model impact
  - False-positive
  - False-negatives
- …

Outline

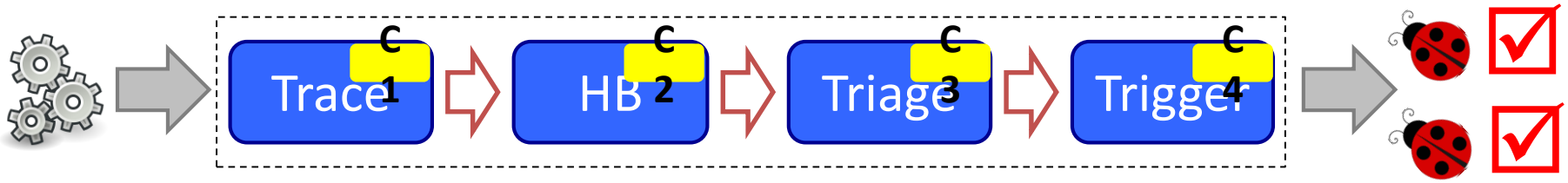
- Motivation
- DCatch Happens-before Model
- DCatch tool
- Evaluation
- Conclusion

Conclusion

- A **HB Model** for distributed systems



- **DCatch** detects DCbugs from **correct runs** with **low false positive** rates.



Thank you!

Q&A