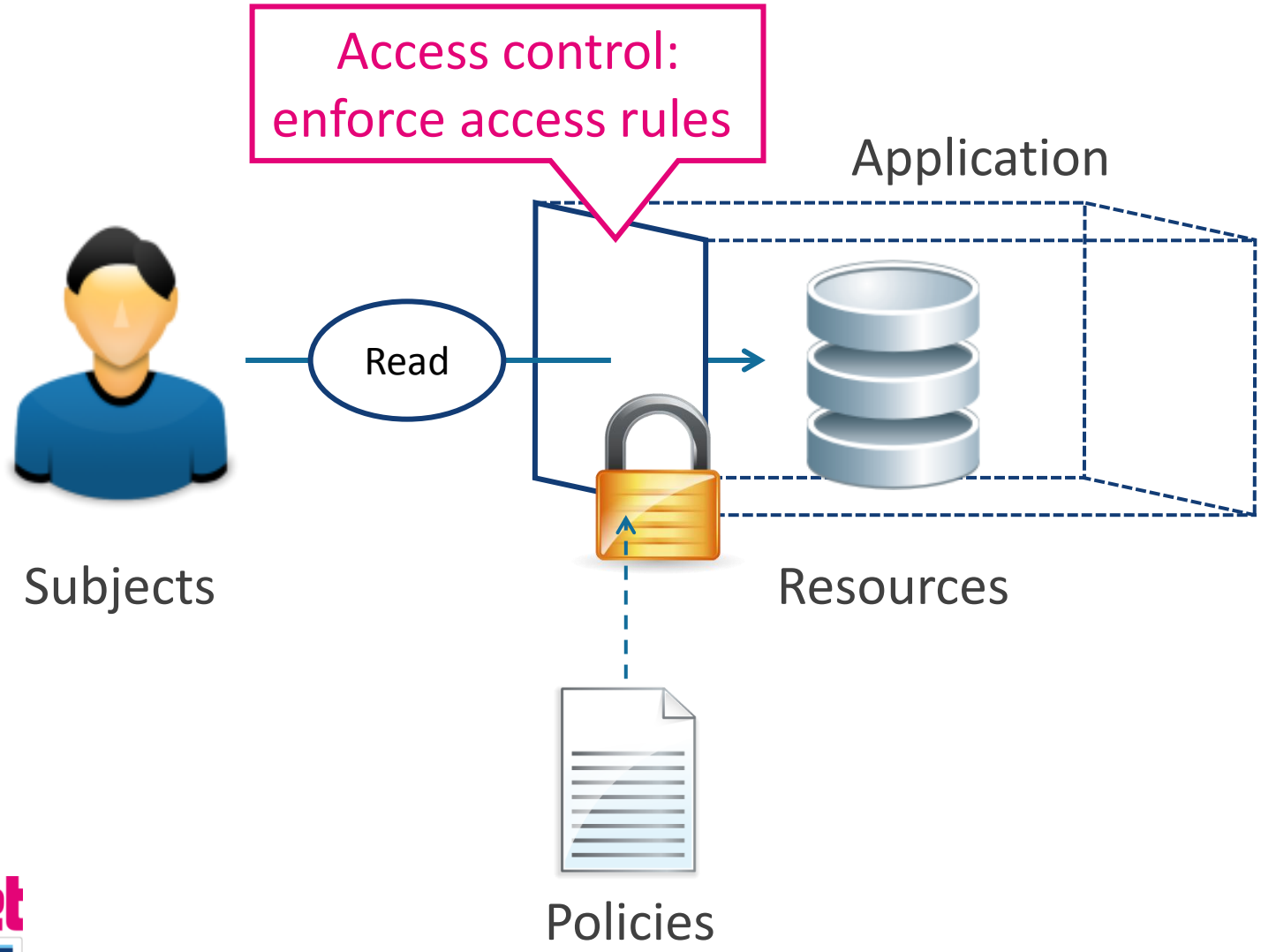
Scalable and Secure Concurrent Evaluation of History-based Access Control Policies

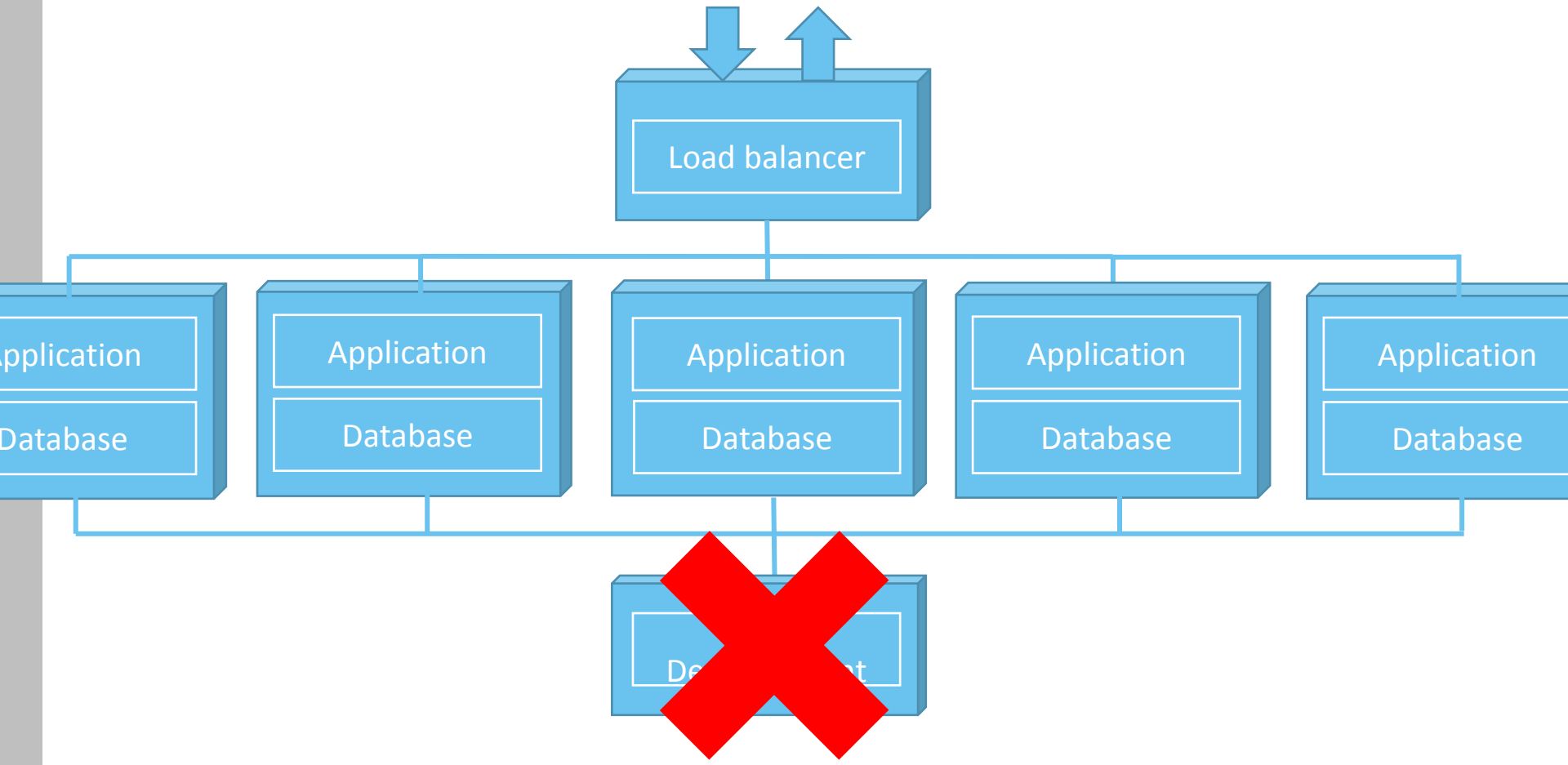
Maarten Decat, Bert Lagaisse, Wouter Joosen
KU Leuven, Belgium
maarten.decat@cs.kuleuven.be

DistriNet
iMinds KU LEUVEN

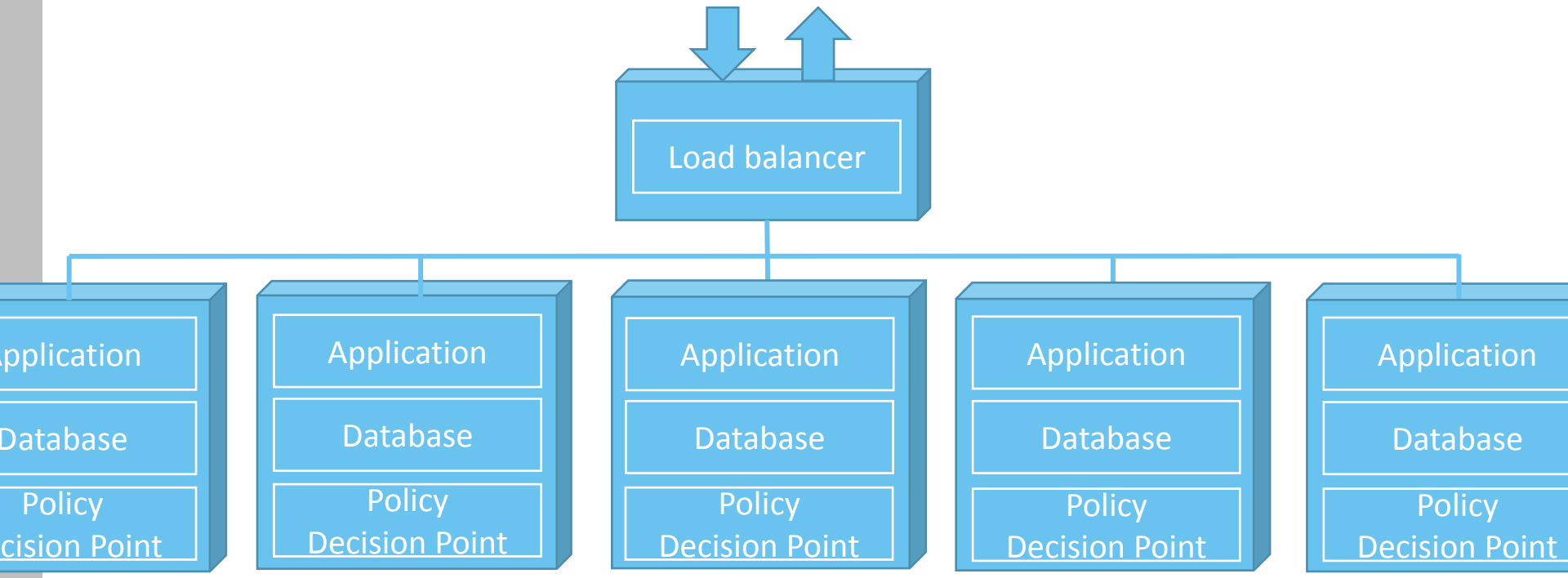
Context


Access control:
enforce access rules
Application
Read
Subjects
Resources
Policies

Context


Load balancer
Application
Database
Application
Database
Application
Database
Application
Database
Application
Database

Context


Load balancer
pplication
Database
Policy
cision Point
Application
Database
Policy
Decision Point
Application
Database
Policy
Decision Point
Application
Database
Policy
Decision Point
Application
Database
Policy
Decision Point

Problem

Problem

Not all policies are read-only:
history-based policies

"a user cannot watch more than 10 movies per month"

"an article cannot be shared more than 5 times"

"if a user has had access to documents of Bank A, he or she is not allowed to access documents of Bank B"
(Dynamic SoD)

DistriNet iMinds KU LEUVEN

Problem

Subject1 -> doc of Bank A

read subj.history *// (= [])*

… // evaluate policy = permit

append "Bank A" to subj.history

return permit

Subject1 -> doc of Bank B

read subj.history *// (= ["Bank A"])*

… // evaluate policy = deny

return deny

Problem

Subject1 -> doc of Bank A

read subj.history *// (= [])*

… // evaluate policy = permit

append “Bank A” to subj.history

return permit

Subject1 -> doc of Bank B

read subj.history *// (= [])*

… // evaluate policy = permit

append “Bank B” to subj.history

return permit

Read-write conflict

Goal

- We need concurrency control
- Possible approach: model policy evaluations as transactions on the underlying database
- However: serial equivalence does not scale
  - E.g., MySQL: single master server for transactions
  - E.g., Cassandra, MongoDB: compare-and-set on single database elements

$\Rightarrow$ we need a domain-specific form of concurrency control to achieve both serial equivalence and scalability

Approach

Approach

When resource.owner **==** “Bank B”,
apply DenyOverrides **to**

Deny if
“Bank A” **in** subject.history

Permit performing
append(“Bank B”, subject.history)

Attributes

Obligations

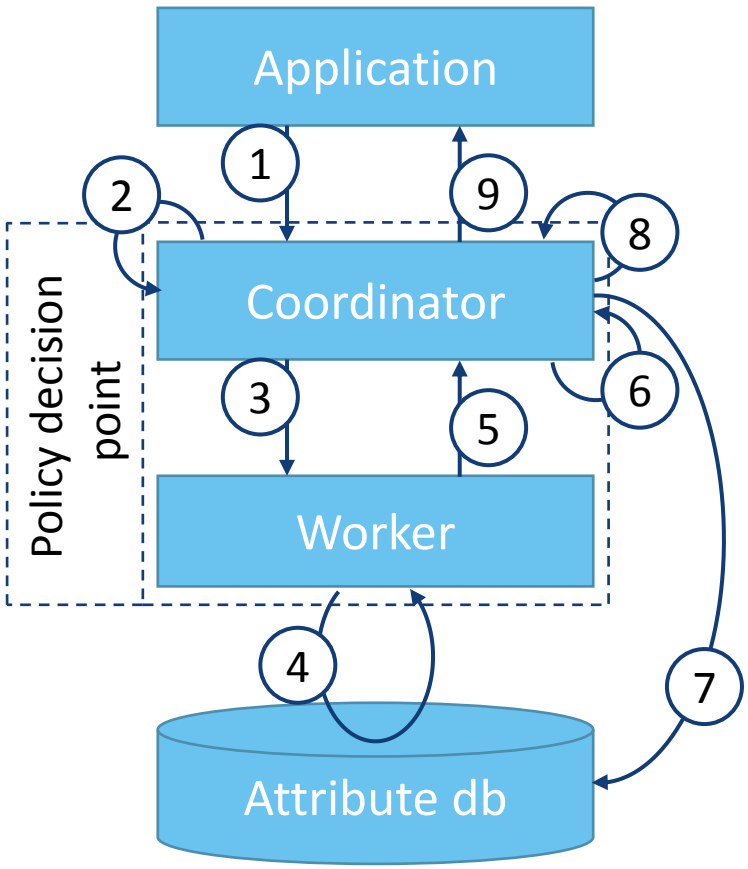
⇒ Goal: detect and contain read-write conflicts for the same attribute in parallel policy evaluations

Possible tactics

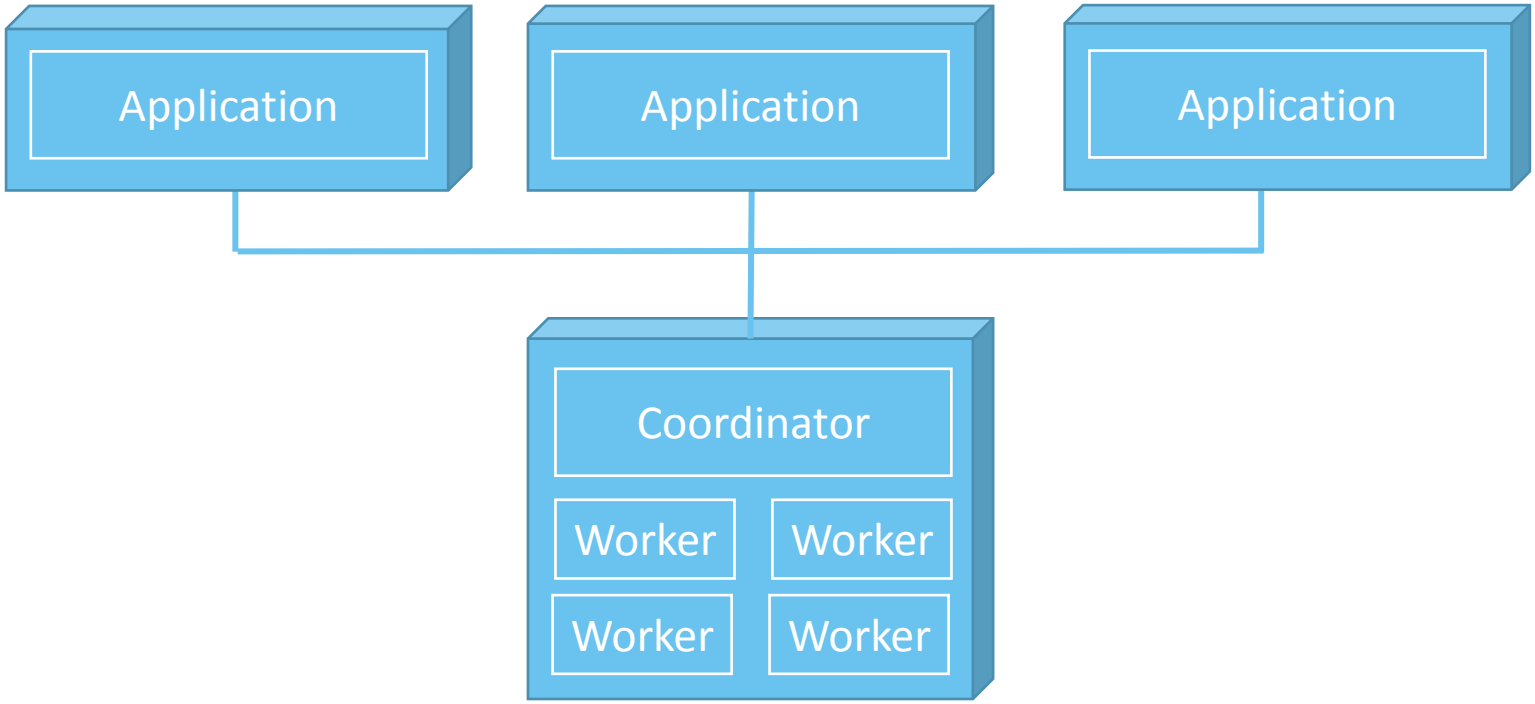
1. Locks
2. (Multi-version) time-stamp ordering
3. Optimistic concurrency control
   - Check for conflicts before committing
   - Roll back any updates in case of conflict
   - Reasons:
     - Fits the structure of a policy evaluation: all attribute updates are performed after all reads
     - Does not block policy evaluations
     - Does not rely on database functionality

Basic approach

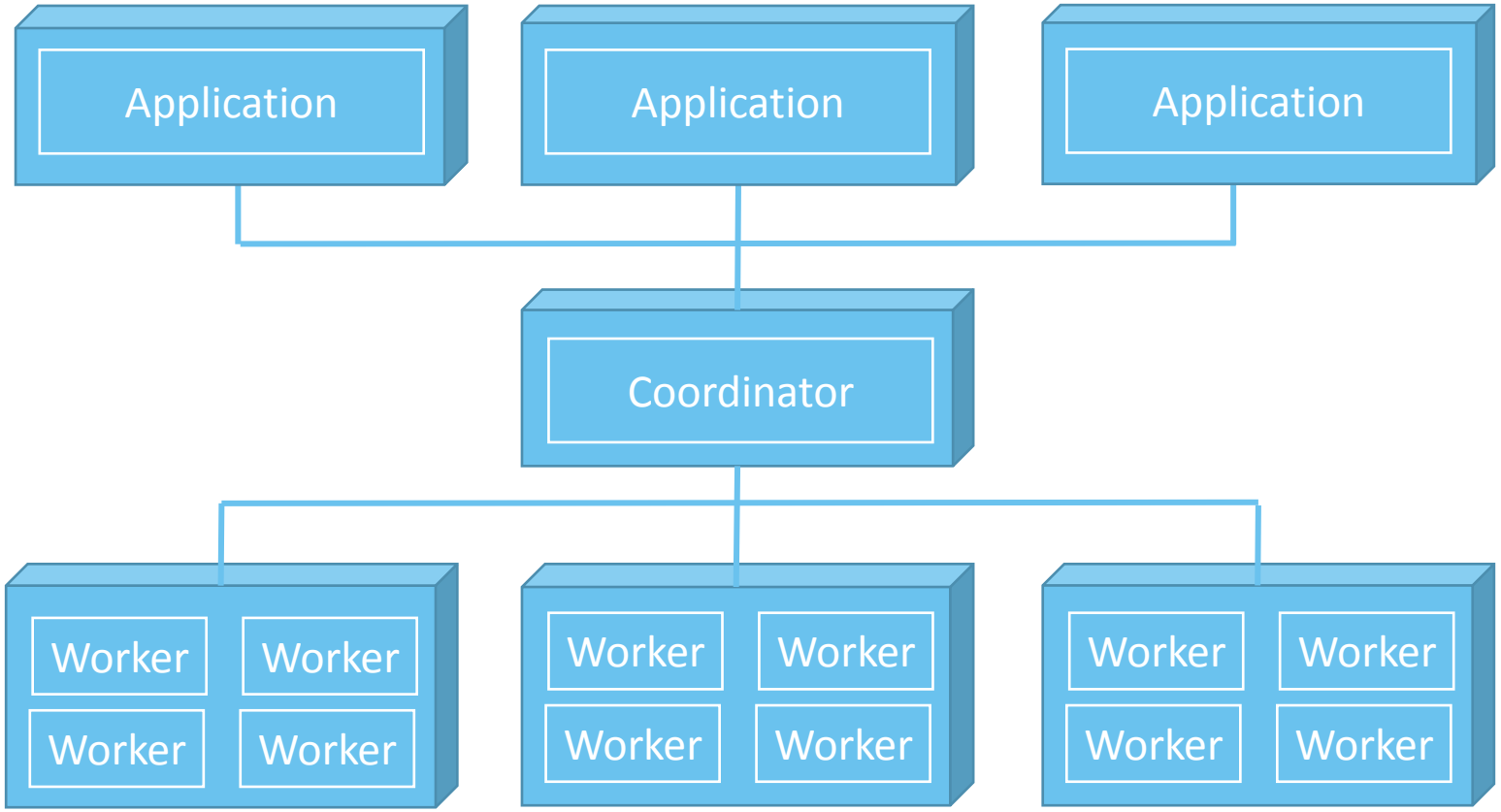
Optimistic concurrency control



Centralized coordinator



Centralized coordinator

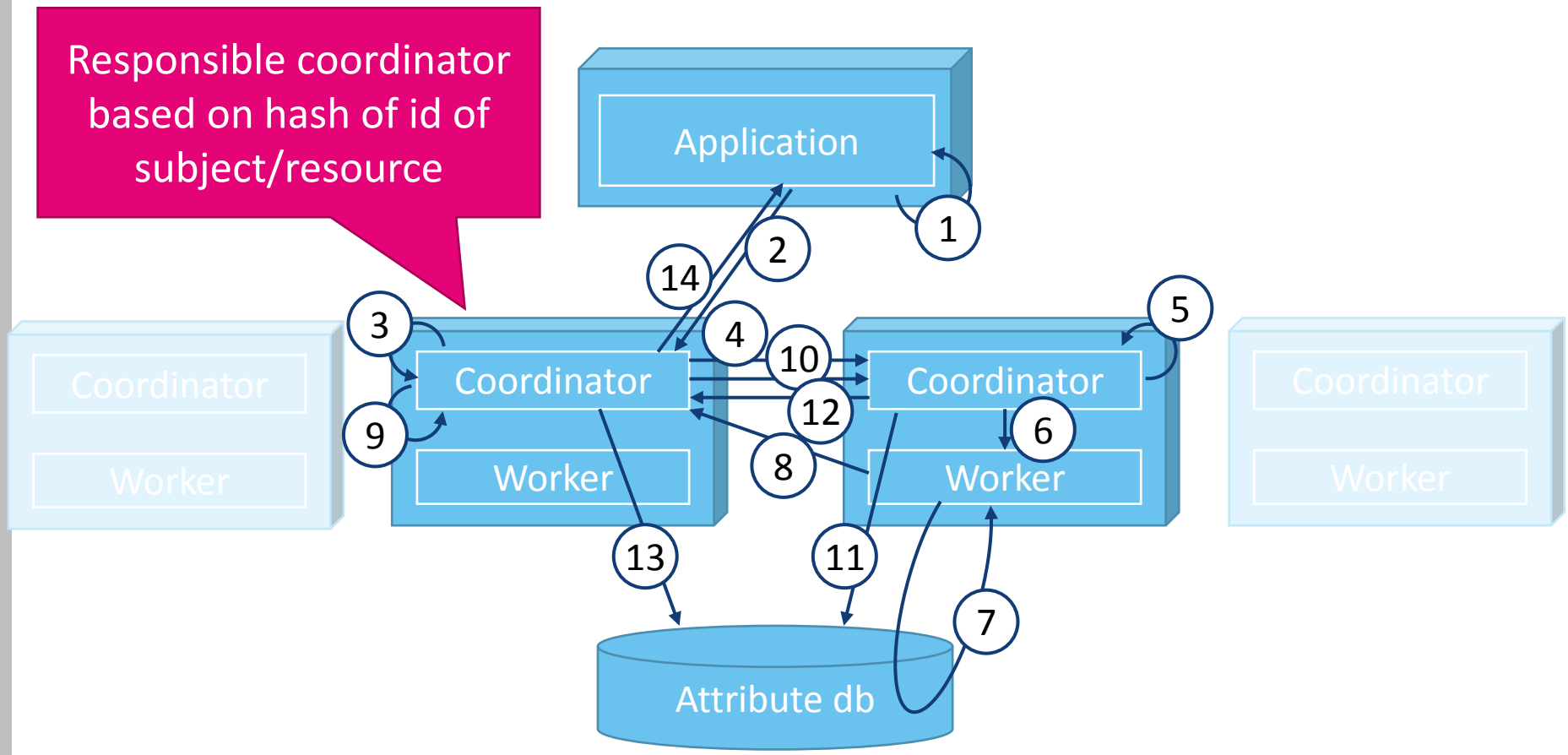


Scalable distributed coordinator

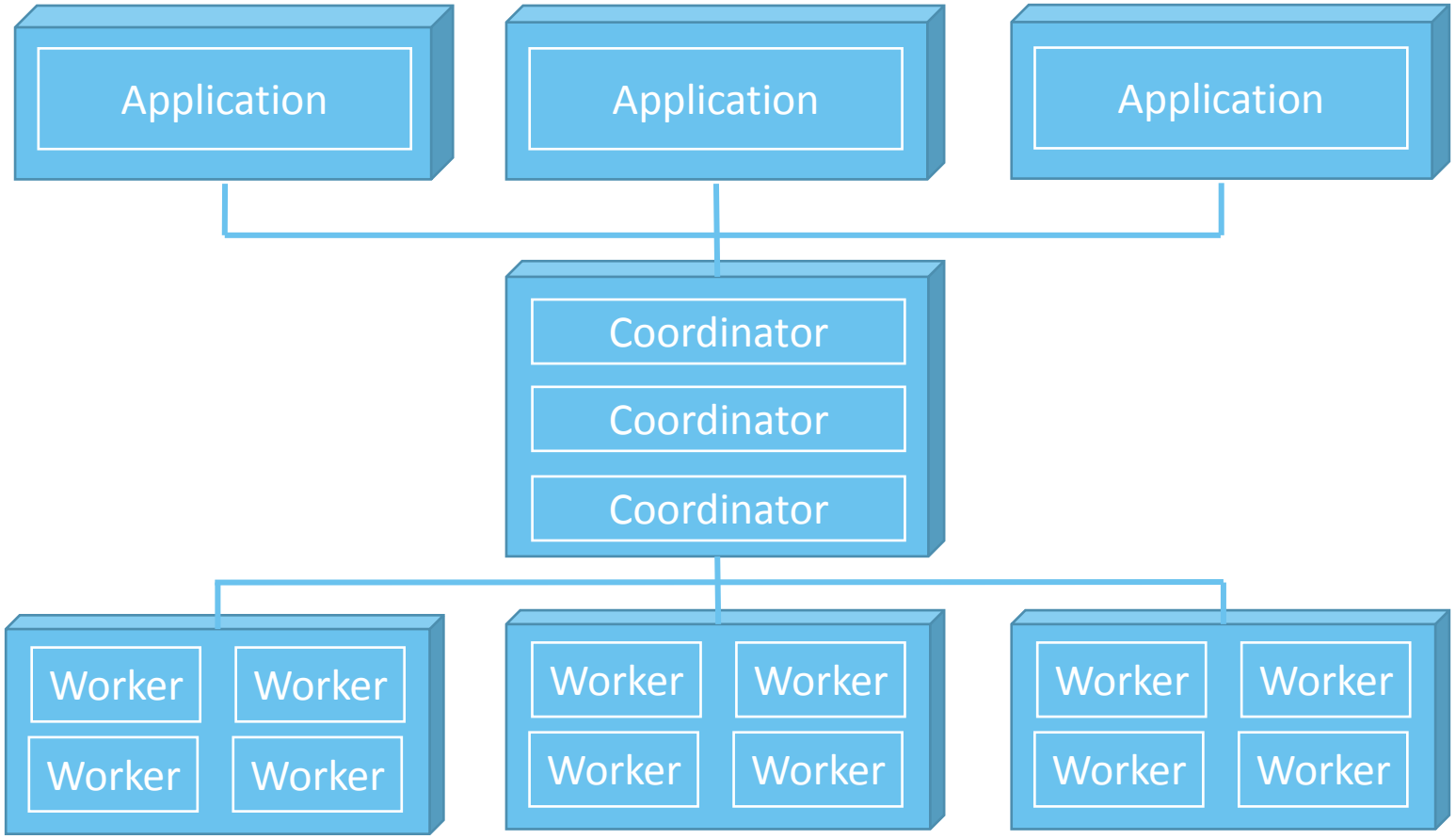
- Observations:
  - Every policy evaluation reasons about exactly 1 subject and 1 resource

⇒ Distribute these two parts over two collaborating coordinators

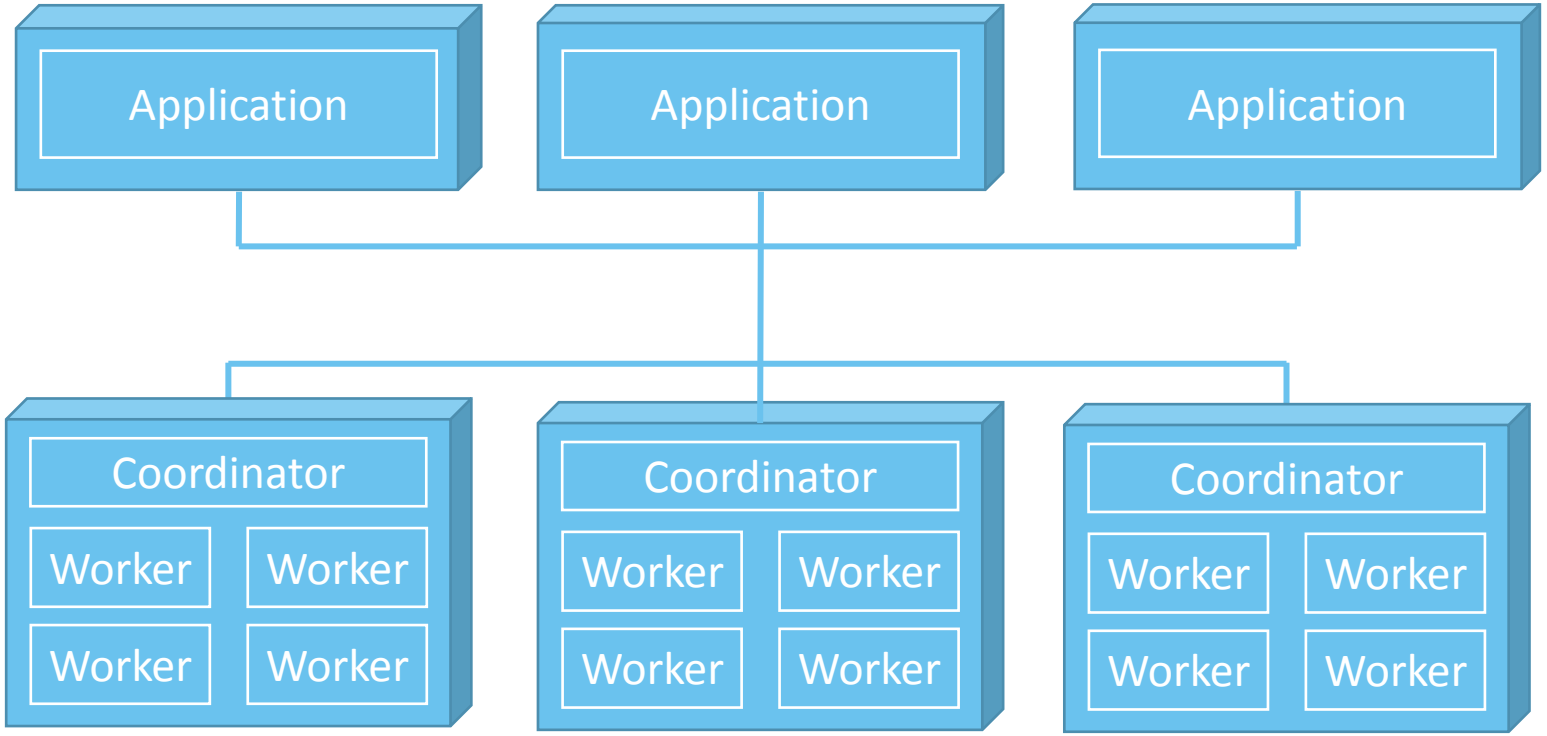
Scalable concurrency control



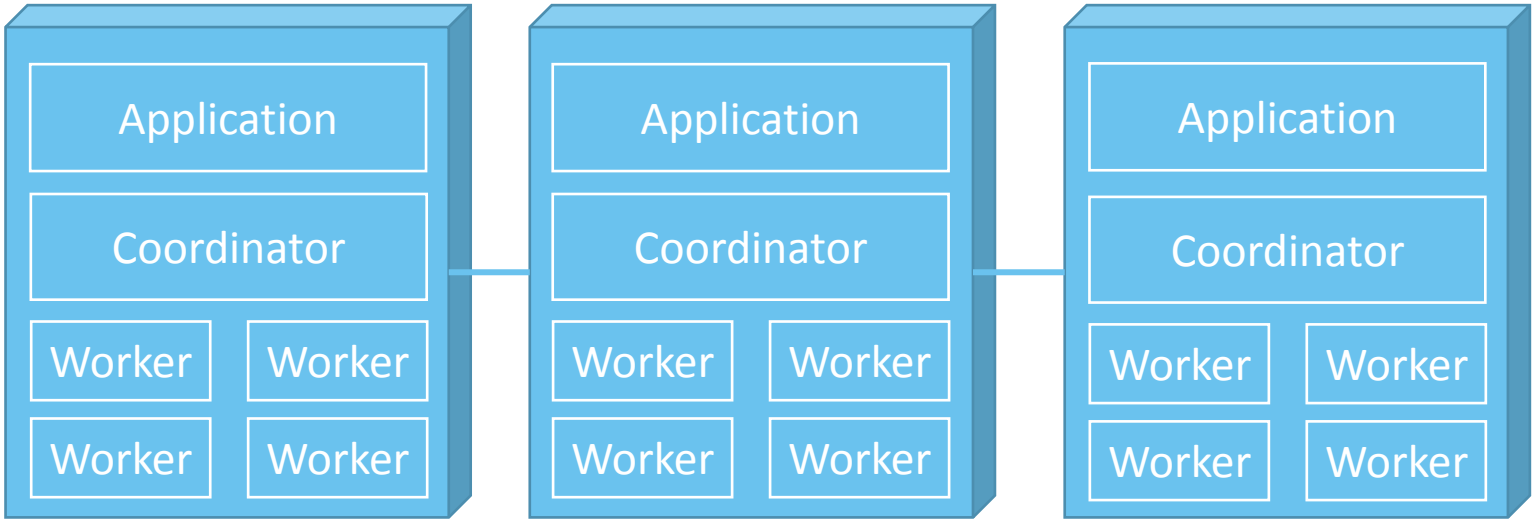
Scalable concurrency control



Scalable concurrency control



Scalable concurrency control

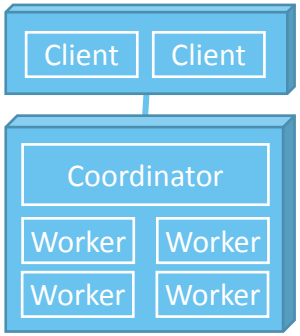


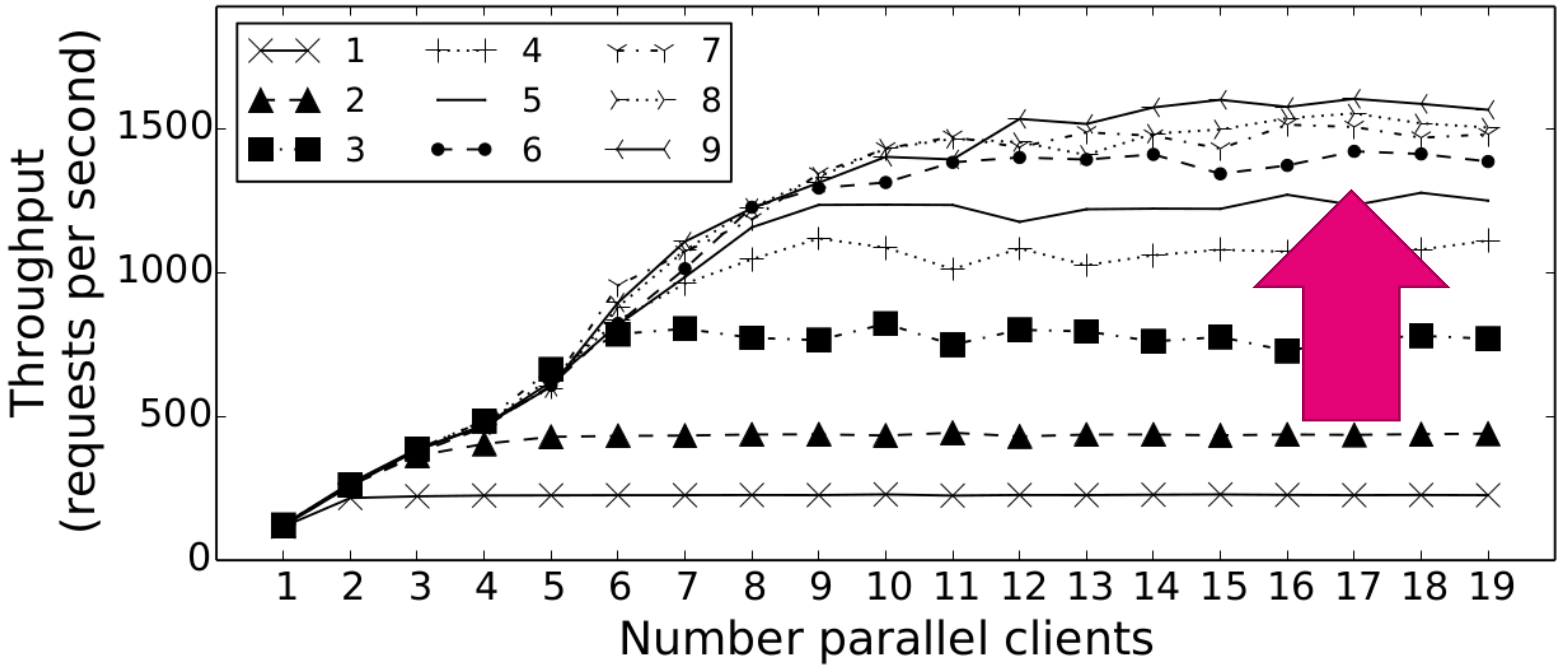
Performance evaluation

Prototype and test set-up

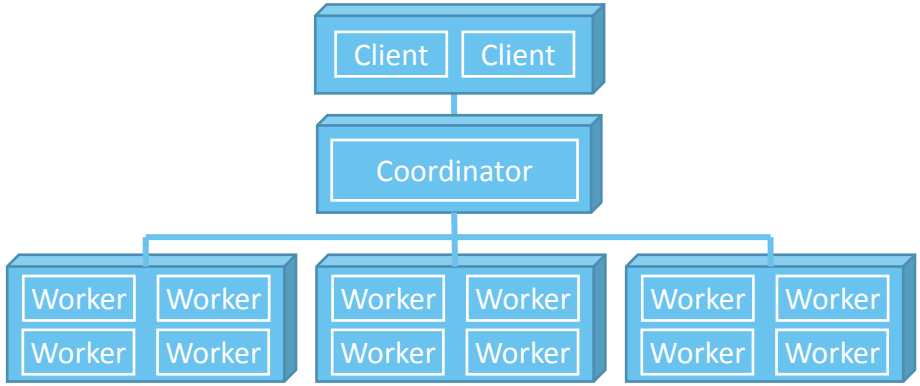
- Prototype:
  - STAPL policy language [https://github.com/stapl-dsl/]
  - Akka actor framework for concurrency and distributed communication
- Test set-up:
  - Run tests on VMs until statistically relevant
  - Employ realistic policy from e-health case study [https://people.cs.kuleuven.be/~maarten.decat/acsac2015/]

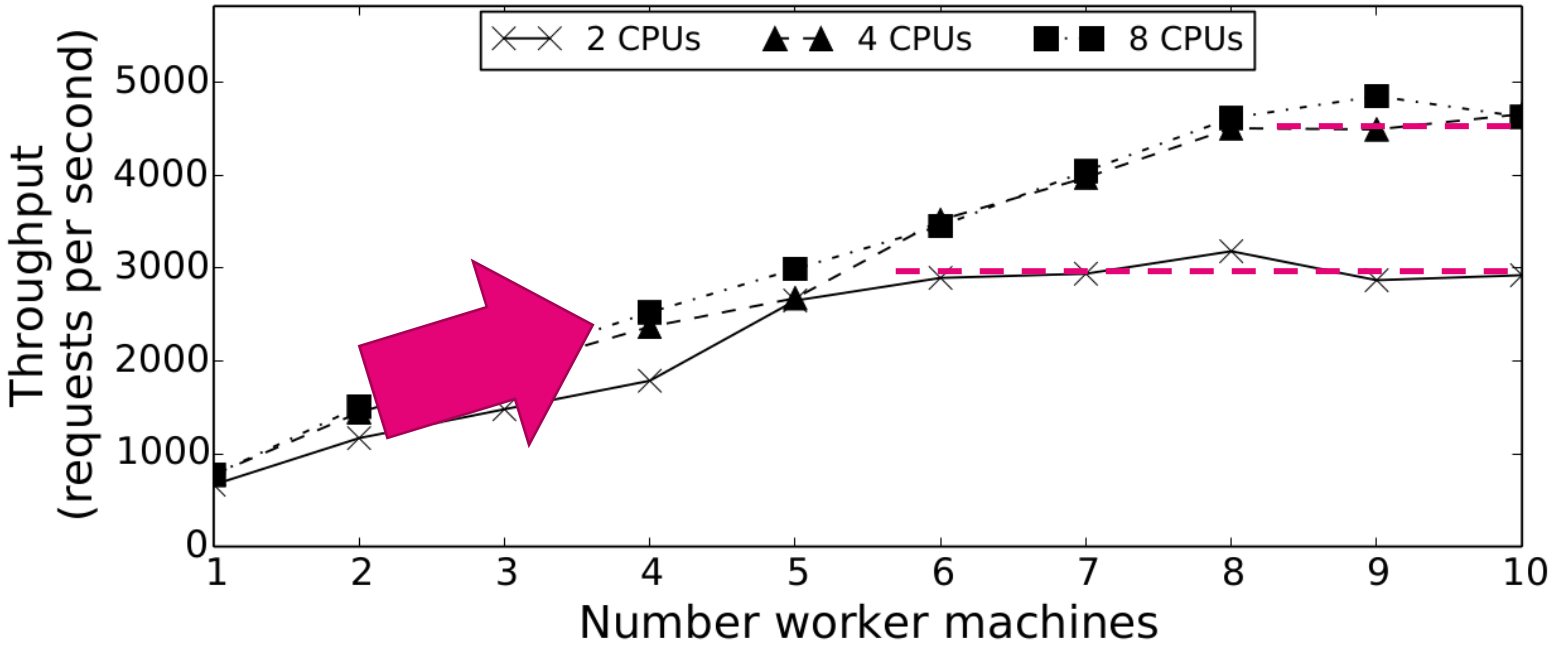
Results


Client
Client
Coordinator
Worker
Worker
Worker
Worker

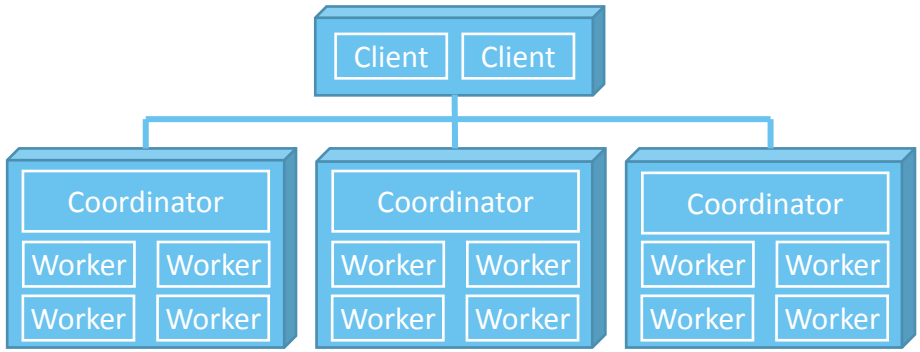

Throughput
(requests per second)
0
500
1000
1500
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Number parallel clients
1
2
3
4
5
6
7
8
9

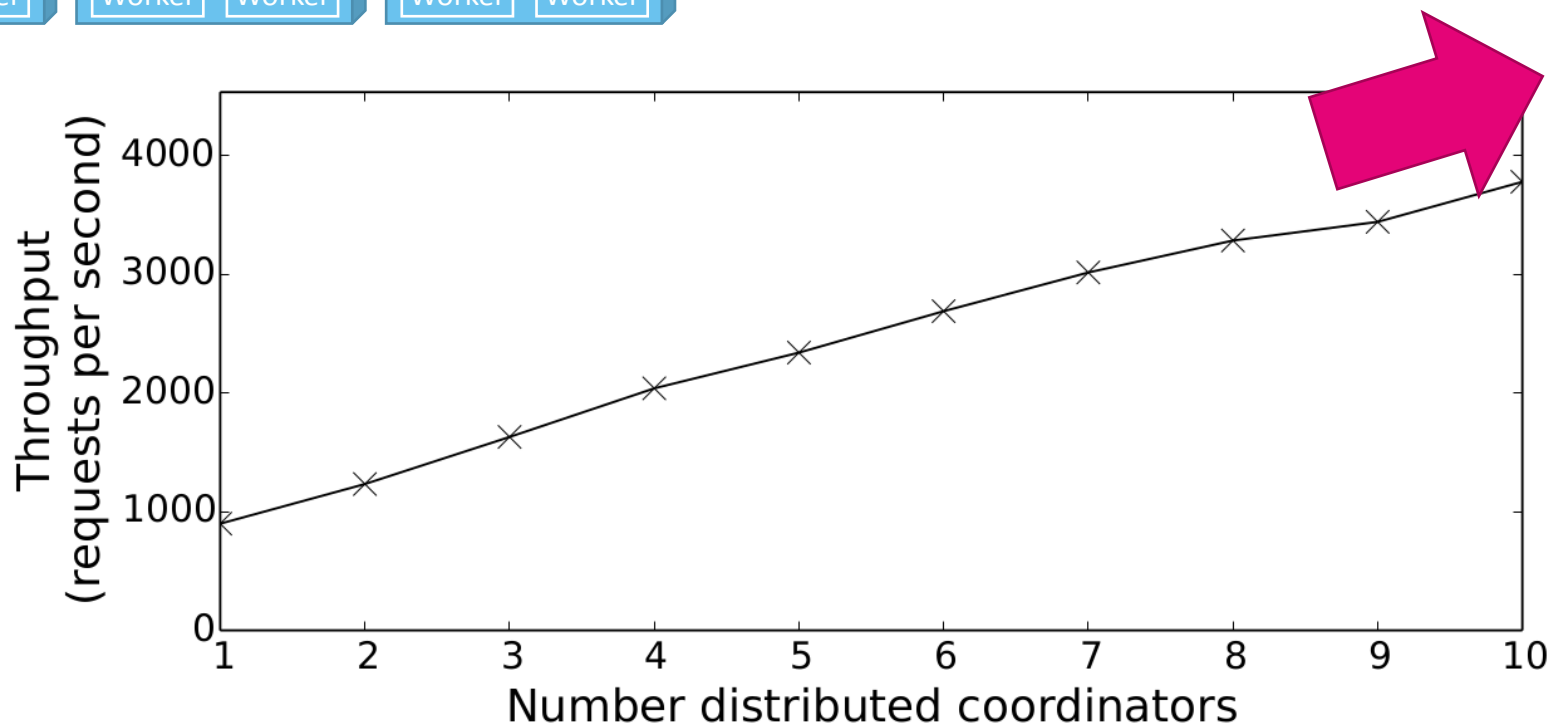
Results


Client
Client
Coordinator
Worker
Worker
Worker
Worker
Worker
Worker
Worker
Worker
Worker
Worker
Worker
Worker

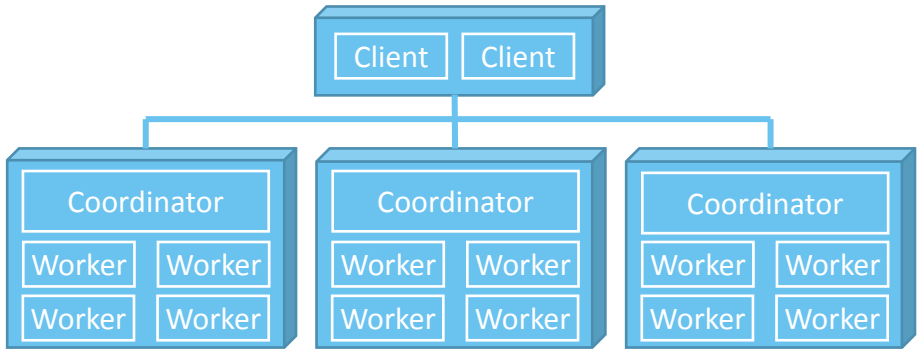

2 CPUs
4 CPUs
8 CPUs
Throughput
(requests per second)
0
1000
2000
3000
4000
5000
1
2
3
4
5
6
7
8
9
10
Number worker machines

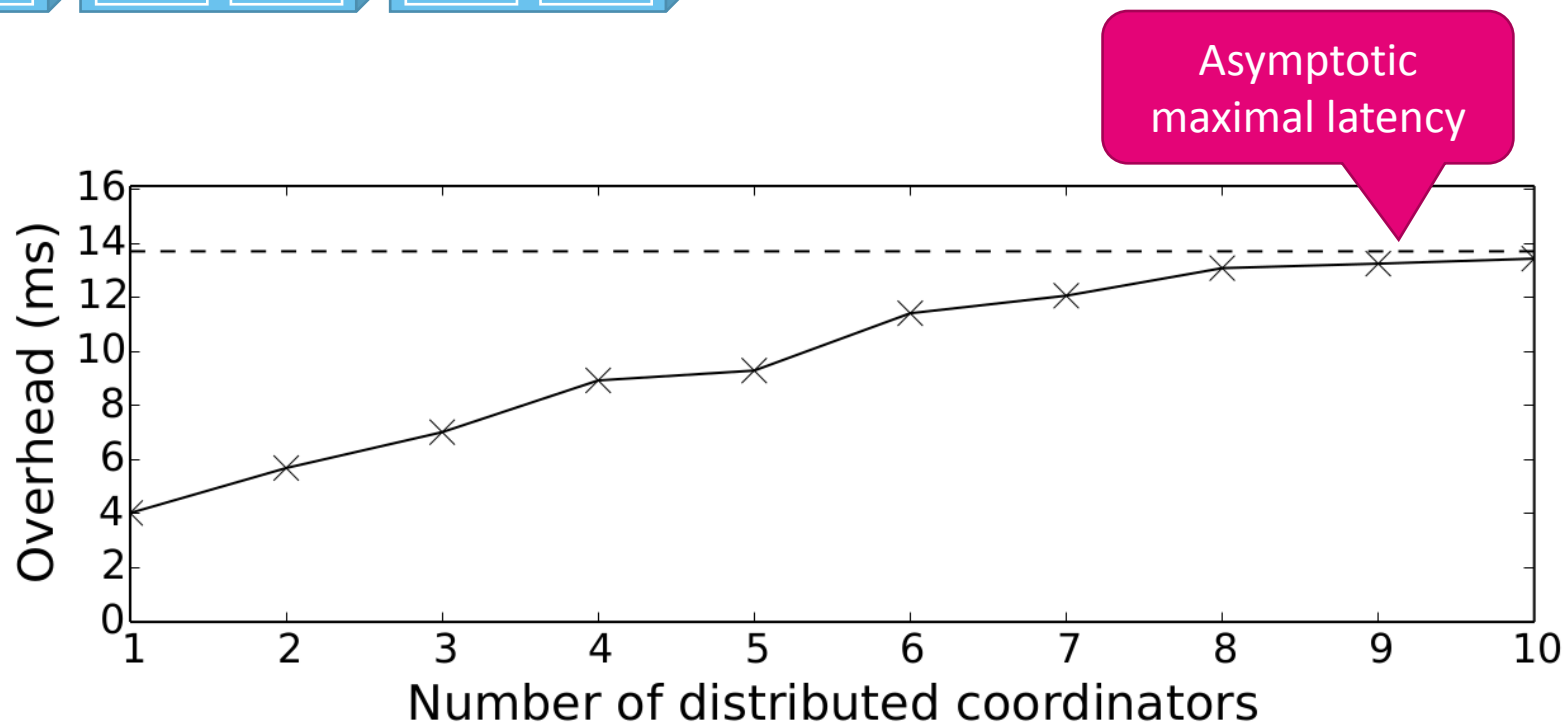
Results


Client
Client
Coordinator
Worker
Worker
Worker
Worker
Coordinator
Worker
Worker
Worker
Worker
Coordinator
Worker
Worker
Worker
Worker


Throughput
(requests per second)
0
1000
2000
3000
4000
1
2
3
4
5
6
7
8
9
10
Number distributed coordinators

Results


Client
Client
Coordinator
Worker
Worker
Worker
Worker
Coordinator
Worker
Worker
Worker
Worker
Coordinator
Worker
Worker
Worker
Worker


Asymptotic maximal latency
Overhead (ms)
0
2
4
6
8
10
12
14
16
1
2
3
4
5
6
7
8
9
10
Number of distributed coordinators

Conclusions

Conclusions

- Goal: applying policy-based access control on large-scale distributed systems
- Focus: address concurrency issues that can arise for history-based access control policies
- Contribution: a domain-specific scheme for concurrency control that:
  - prevents incorrect decisions
  - can scale to a large number of machines
  - introduces a limited and asymptotically bounded latency overhead

Scalable and Secure Concurrent Evaluation of History-based Access Control Policies

Maarten Decat, Bert Lagaisse, Wouter Joosen
KU Leuven, Belgium
maarten.decat@cs.kuleuven.be

DistriNet
iMinds KU LEUVEN