Authorization Recycling in RBAC Systems

Qiang Wei[1], Jason Crampton[2], Konstantin Beznosov[1], Matei Ripeanu[1]

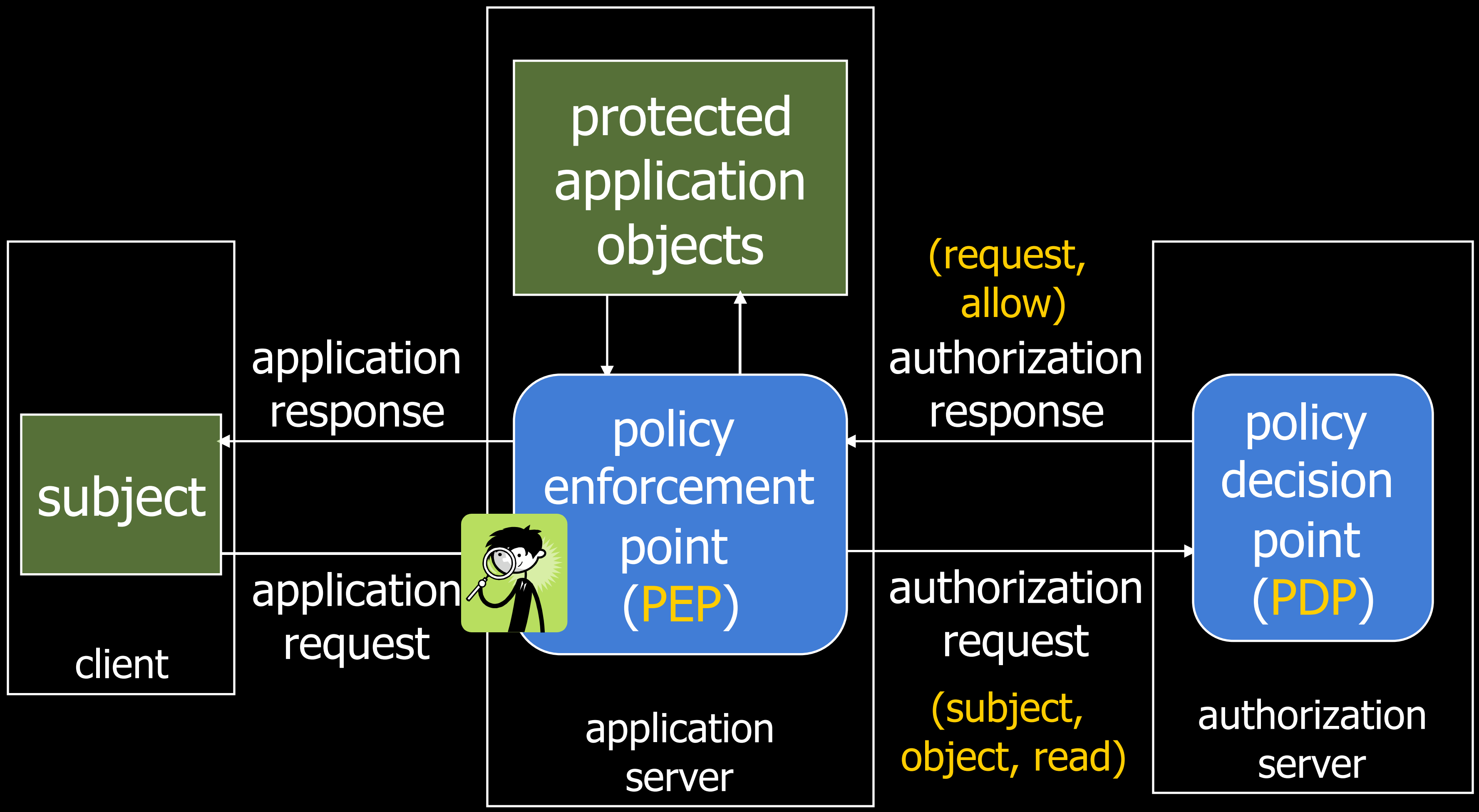
[1]Laboratory for Education and Research in Secure Systems Engineering (**LERSSE**),
University of British Columbia

[2]Information Security Group,
Royal Holloway,
University of London

outline

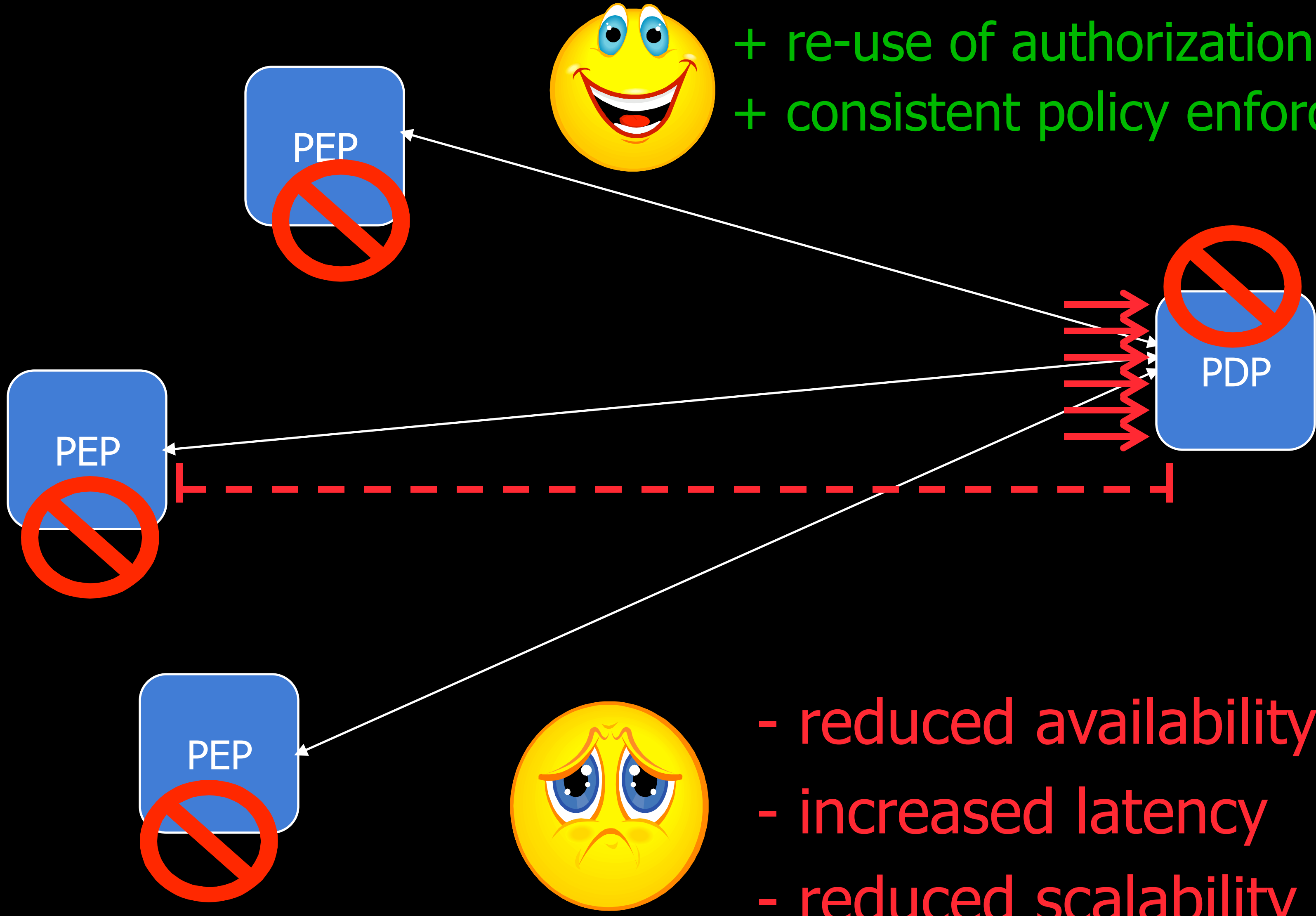
- the overview
  - authorization architecture
  - motivation
  - recycling approach
- recycling algorithms
- experimental evaluations
- summary & future work

a typical authorization architecture



- also known as request-response paradigm
- applied by IBM Access Manager, Entrust GetAccess, CA SiteMinder, etc.

motivations



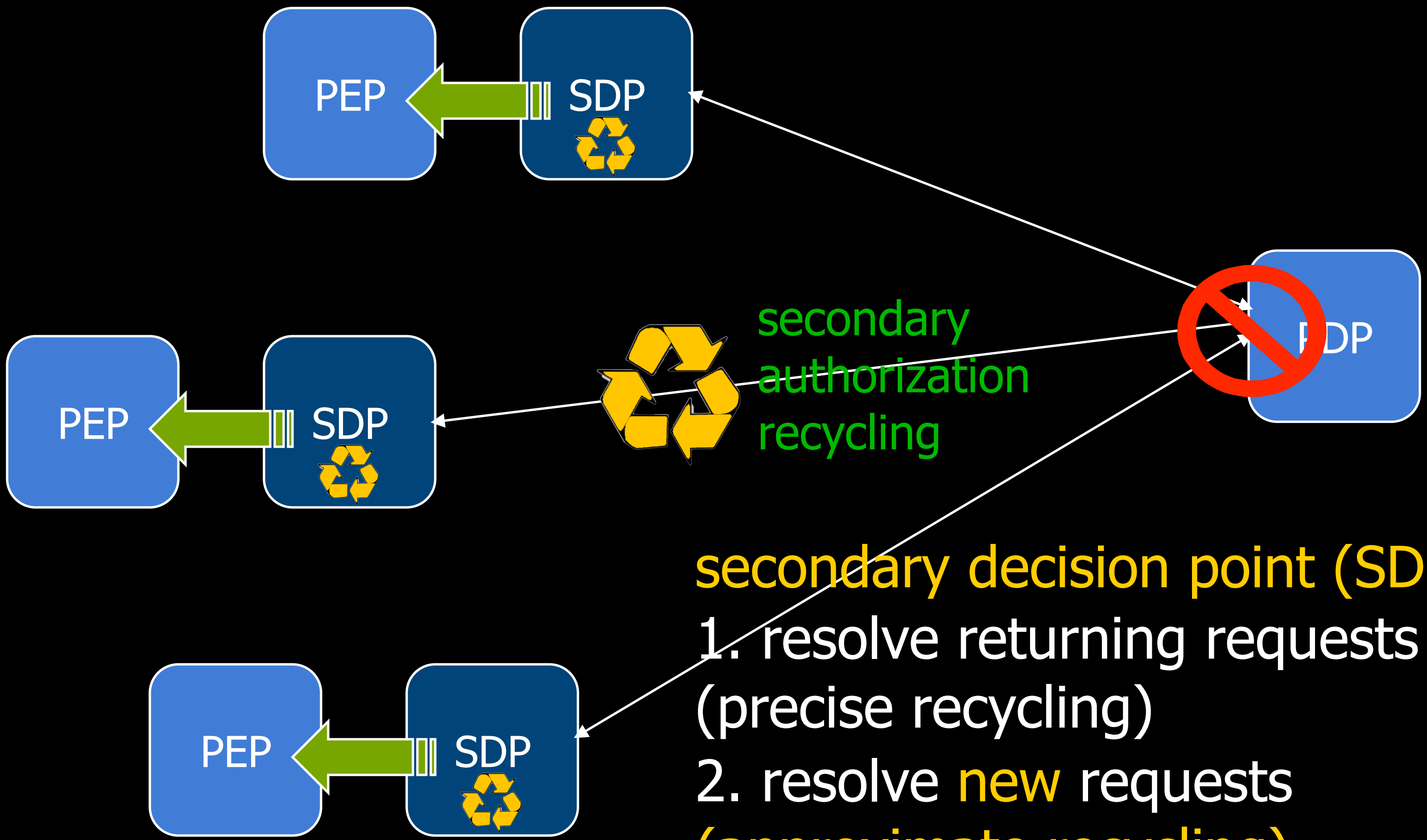
+ re-use of authorization logic
+ consistent policy enforcement

- reduced availability
- increased latency
- reduced scalability

existing approaches

- fault-tolerance by replication/redundancy
  - \+ improve availability
  - \- latency remains unchanged
  - \- require specialized OS/middleware
  - \- poorly scale on large populations
- caching previous authorizations
  - \+ simple, inexpensive
  - \+ improves performance & availability
  - \- serves only requests that have been issued before (precise recycling)

secondary and approximate authorization model (SAAM)[1]



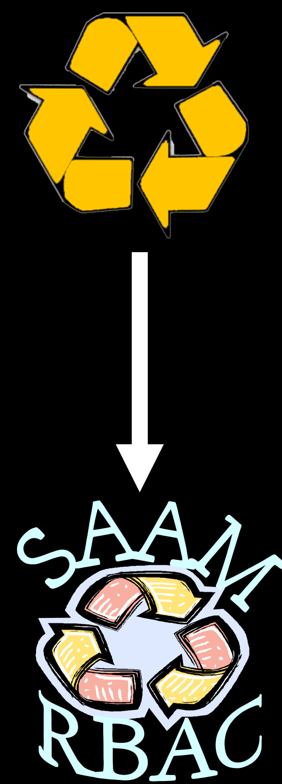
secondary decision point (SDP)

1. resolve returning requests (precise recycling)
2. resolve new requests (approximate recycling)

[1]J. Crampton, W. Leung and K. Beznosov, "The Secondary and Approximate Authorization Model and its Application to Bell-LaPadula Policies," in the *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT)*, Lake Tahoe, California, USA, 7-9 June, 2006.

SAAM$_{RBAC}$

- SAAM
  - only an abstract model
  - a specific SAAM recycling algorithm is needed for each access control model
- SAAM$_{RBAC}$
  - apply SAAM to role-based access control (RBAC) model
  - develop recycling algorithms specifically for RBAC



outline

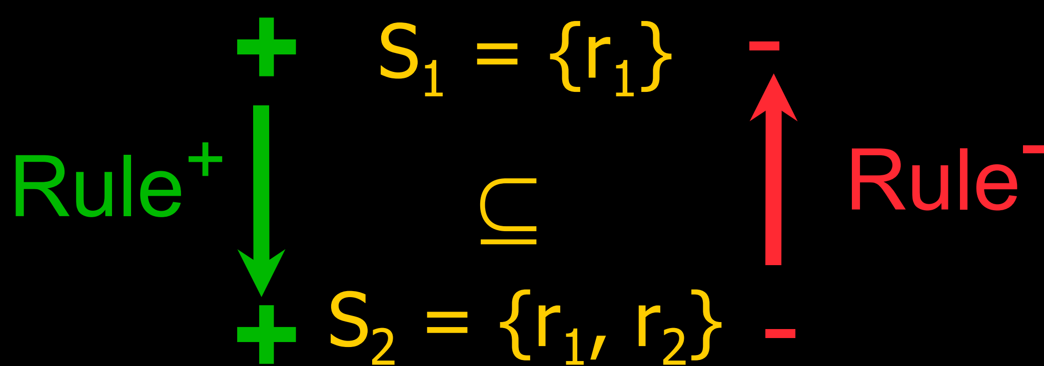
- the overview
- recycling algorithms
- experimental evaluations
- summary & future work

terminology

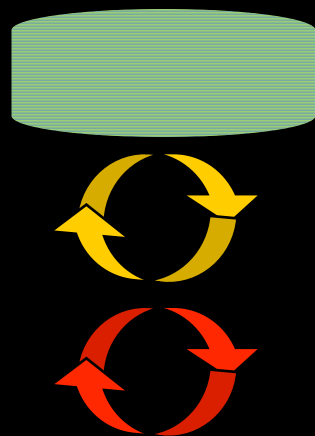
- request: issued by a subject *s* for a permission *p*
  - request=(s,p)
- ±: denotes the decision to a request
  - an allow response: +(s,p)
  - a deny response: –(s,p)
- subject: modeled as the set of roles r activated in a session
  - $s= \{r_1, r_2, r_3\}$

inference rules

- **Rule^{+}**: if +(s,p) and s′ ⊆ s, then request (s',p) should also be allowed
- **Rule^{-}**: if -(s,p) and s′ ⊆ s, then request (s',p) should also be denied



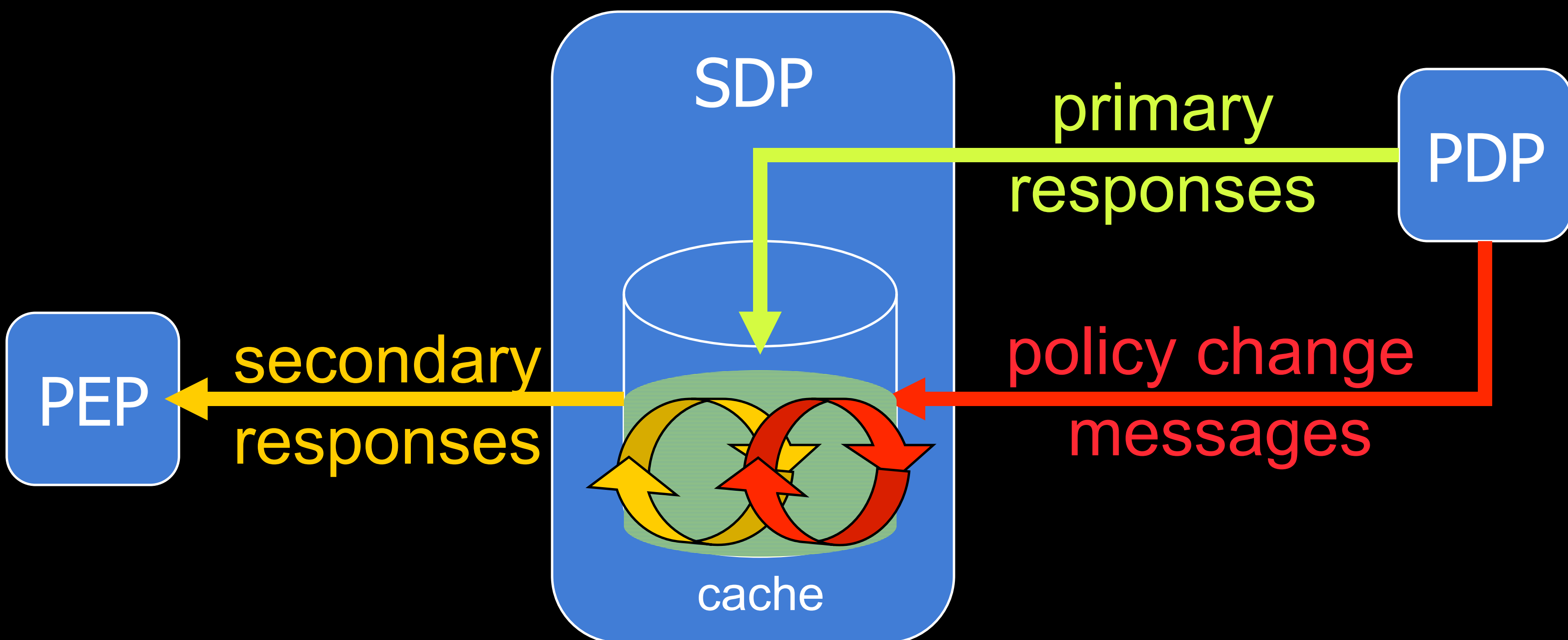
SAAM$_{RBAC}$ recycling algorithms



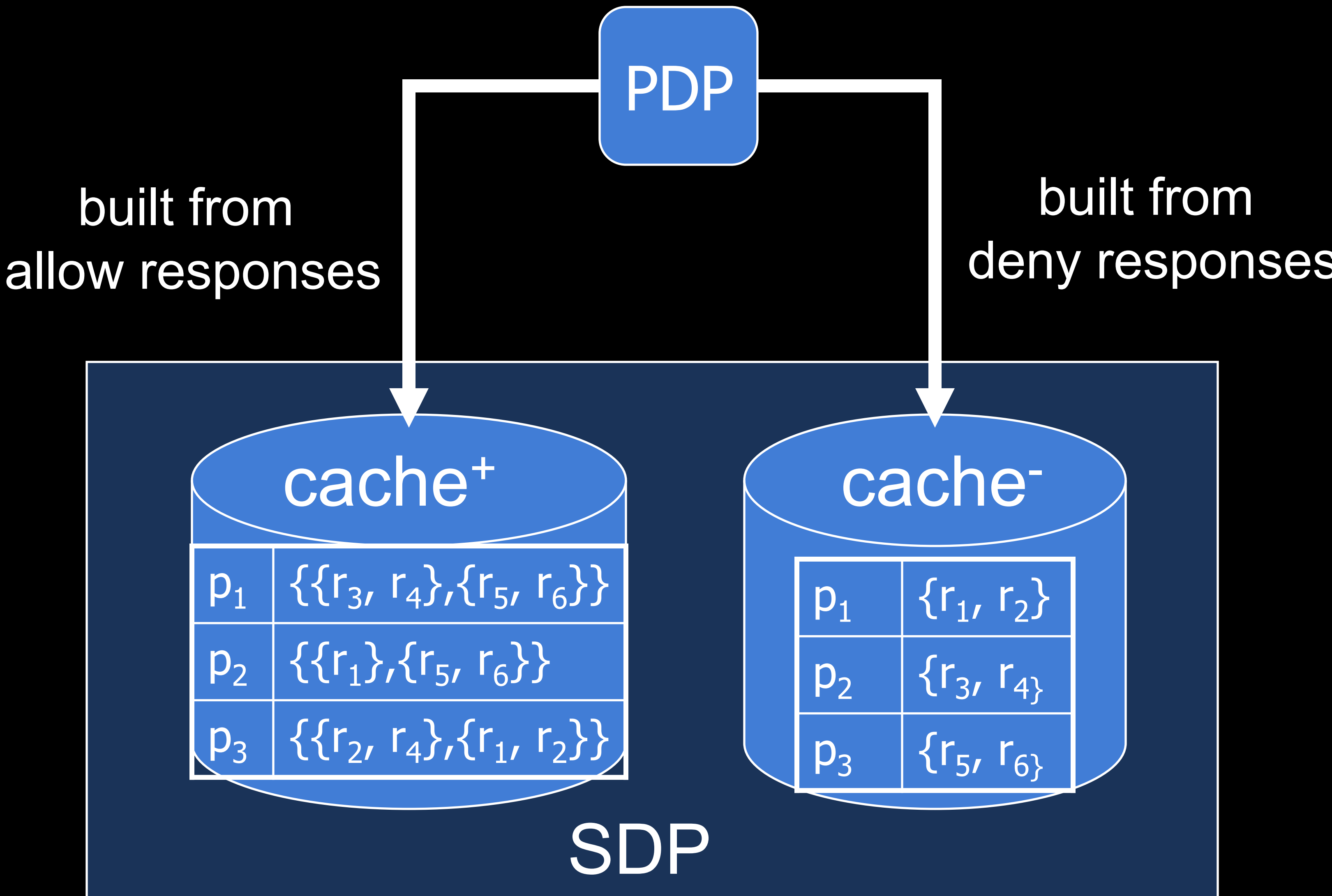
cache construction √

decision √

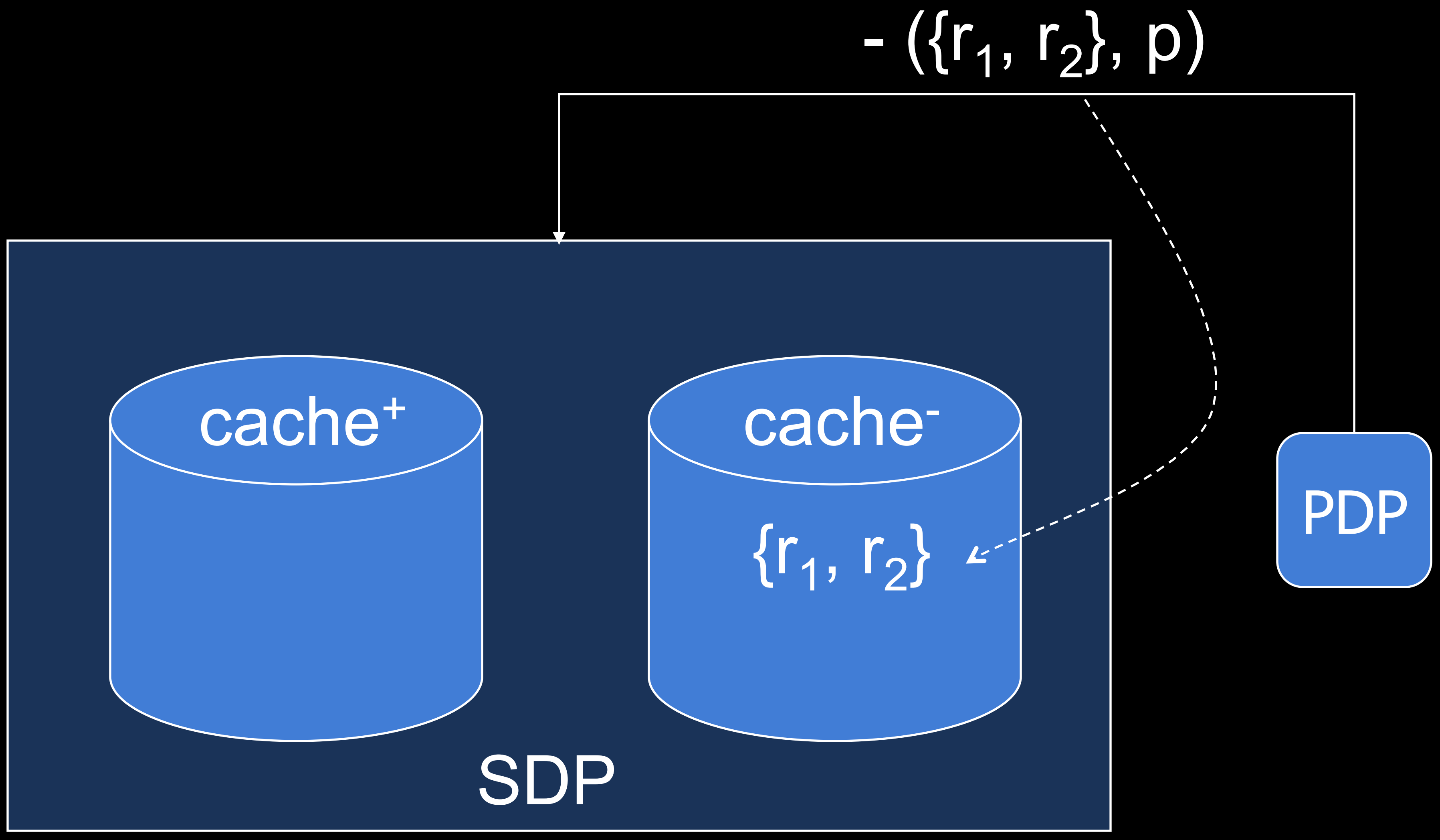
cache update



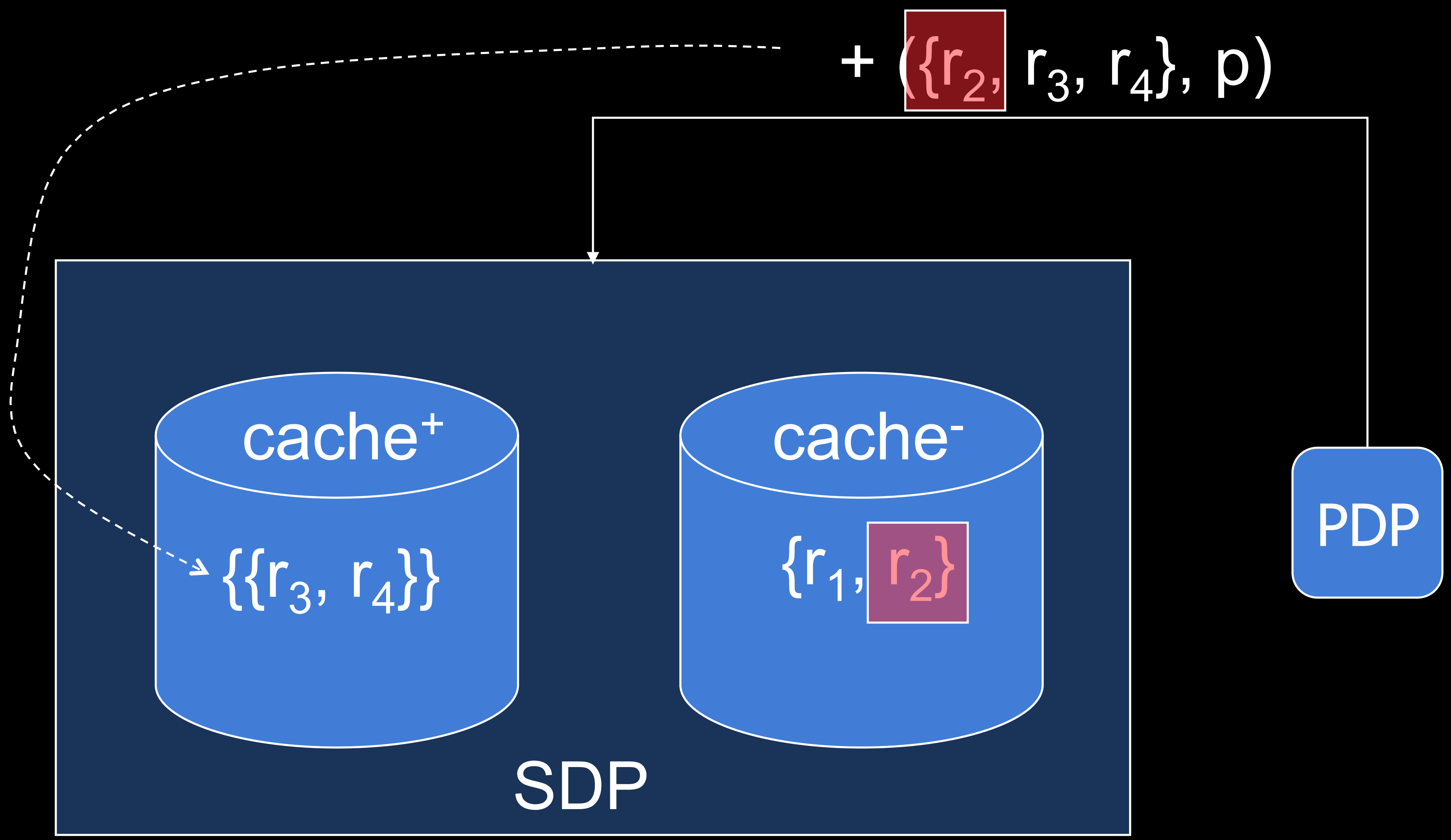
$cache^+$ and $cache^-$



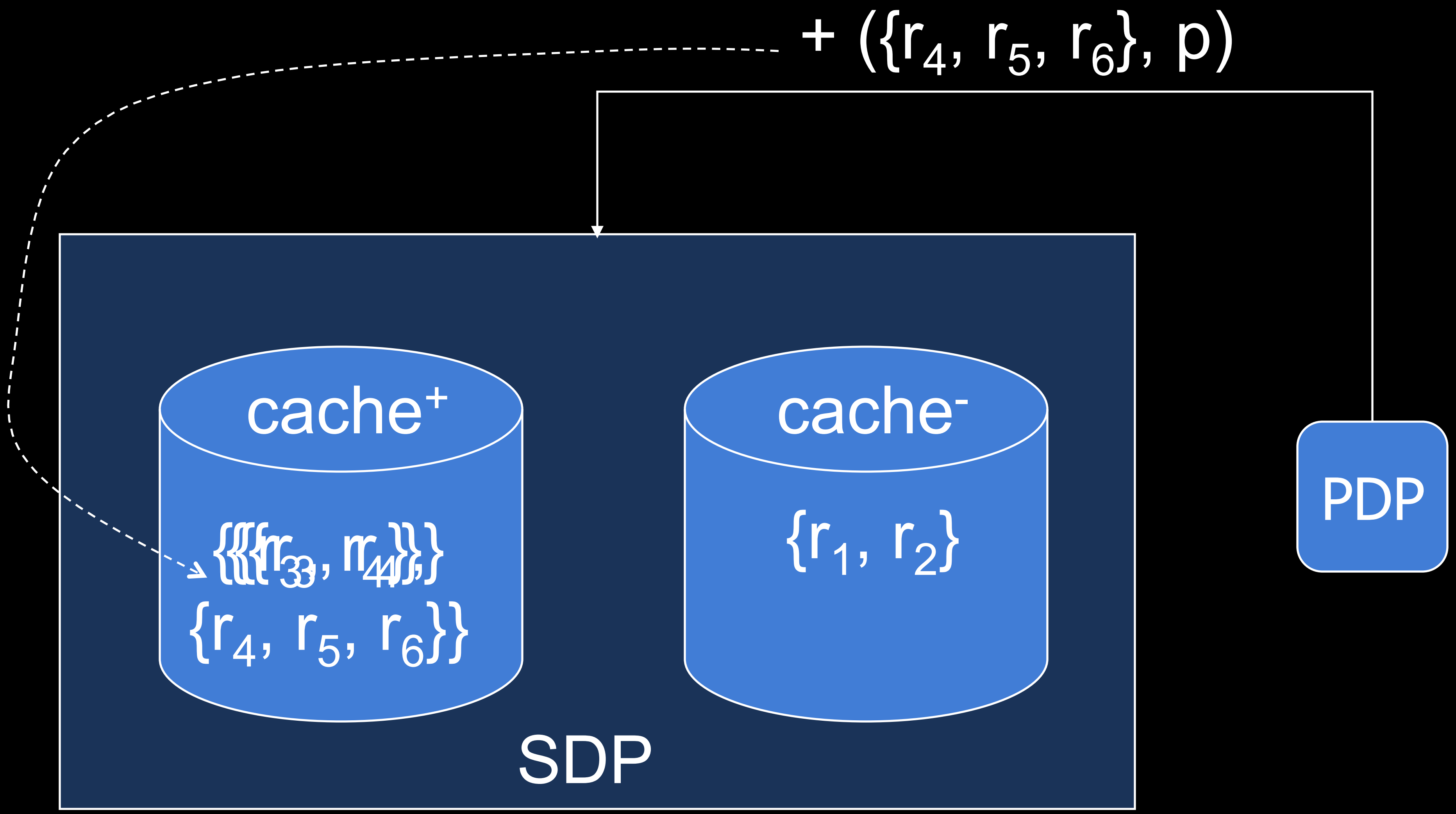
example: SDP receives 1st deny response



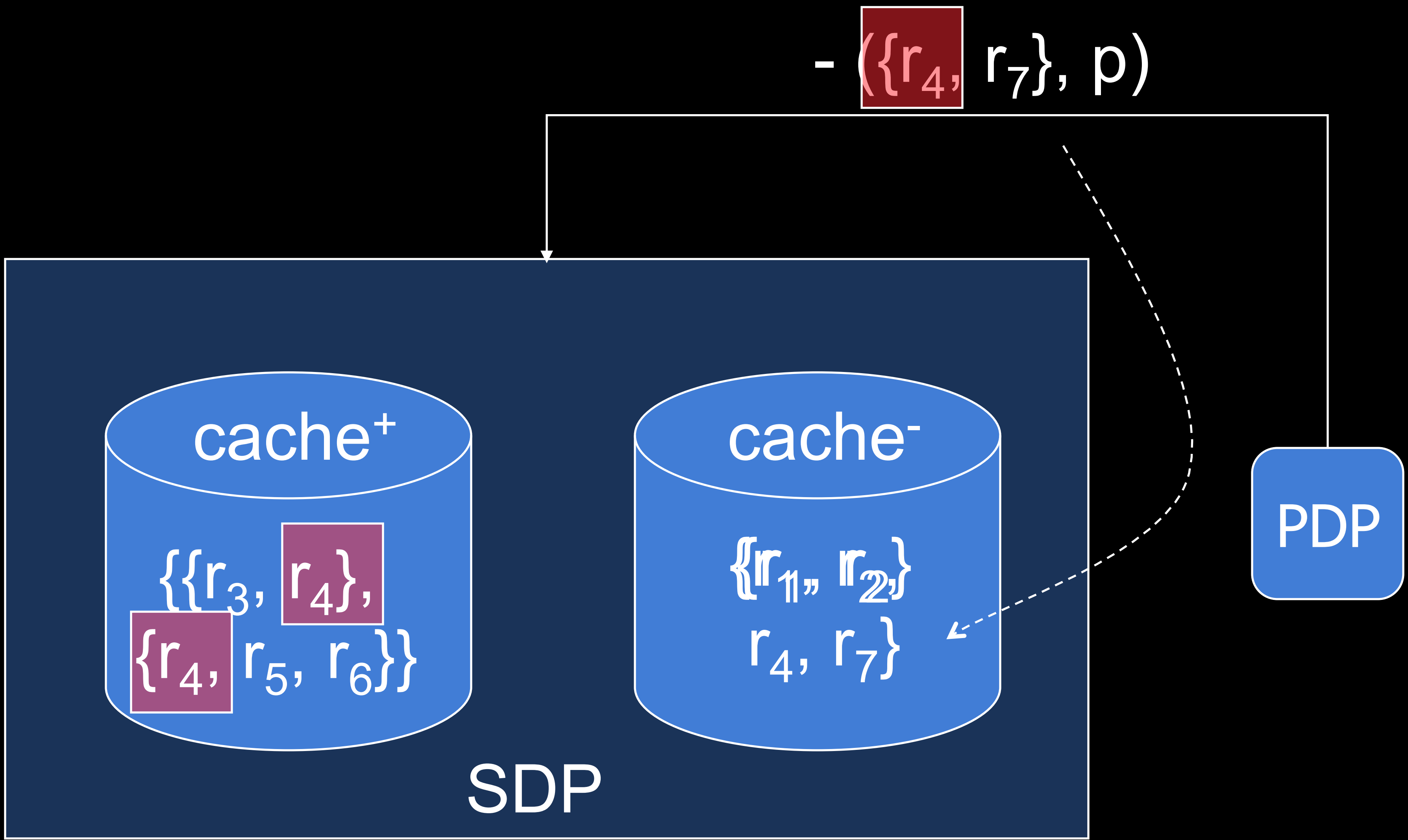
example: SDP receives 1st allow response



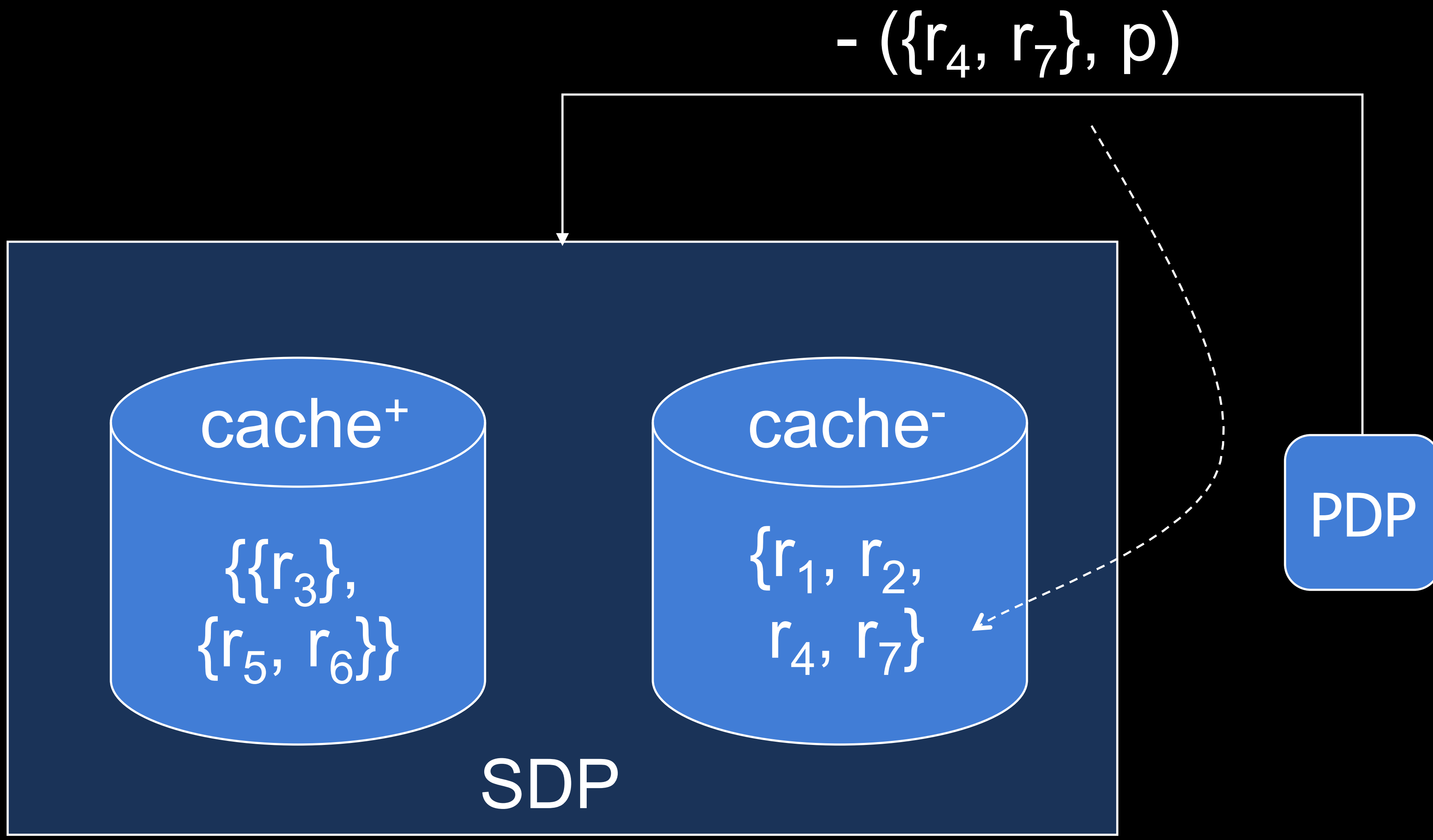
example: SDP receives 2nd allow response



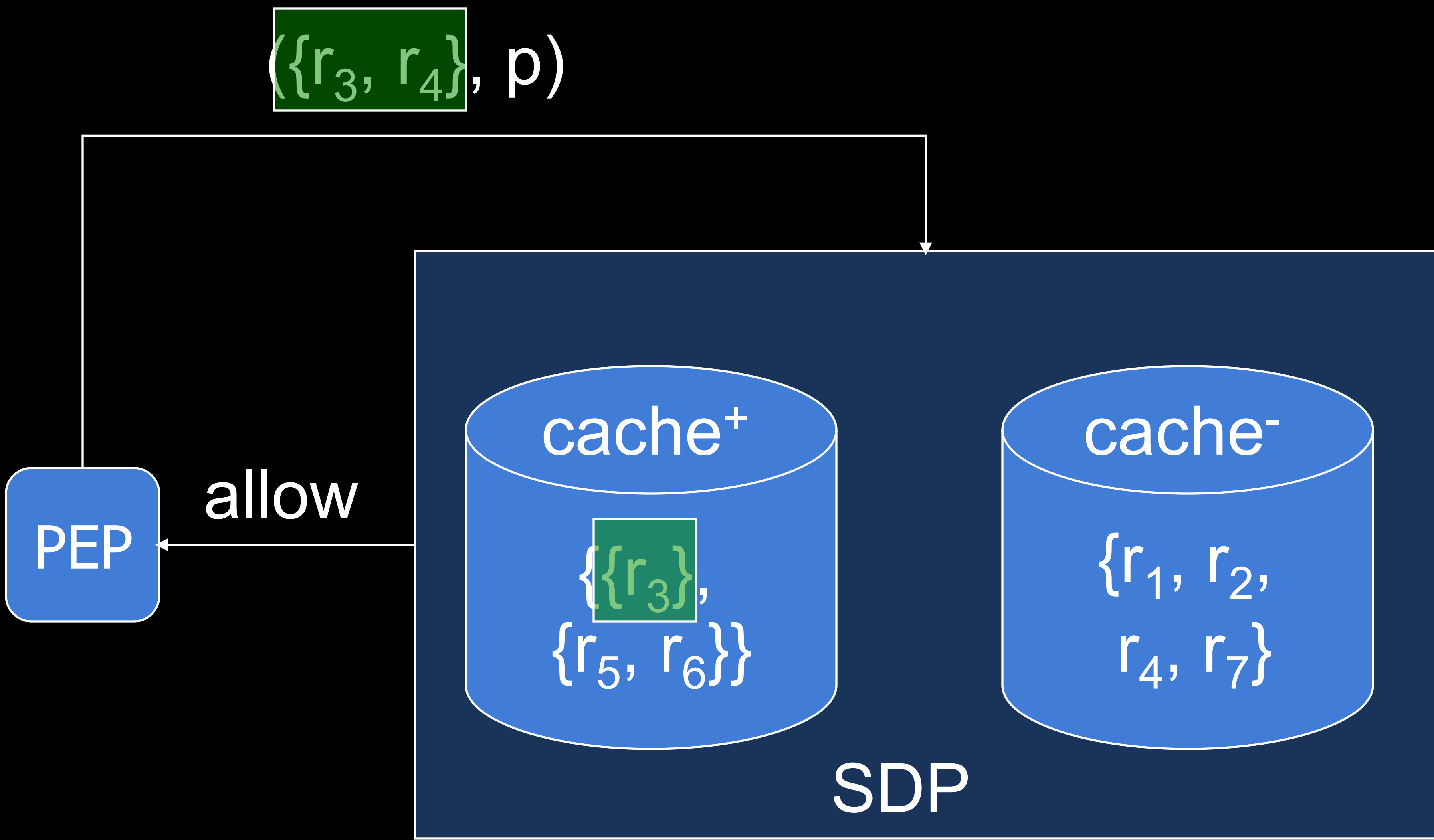
example: SDP receives 2nd deny response



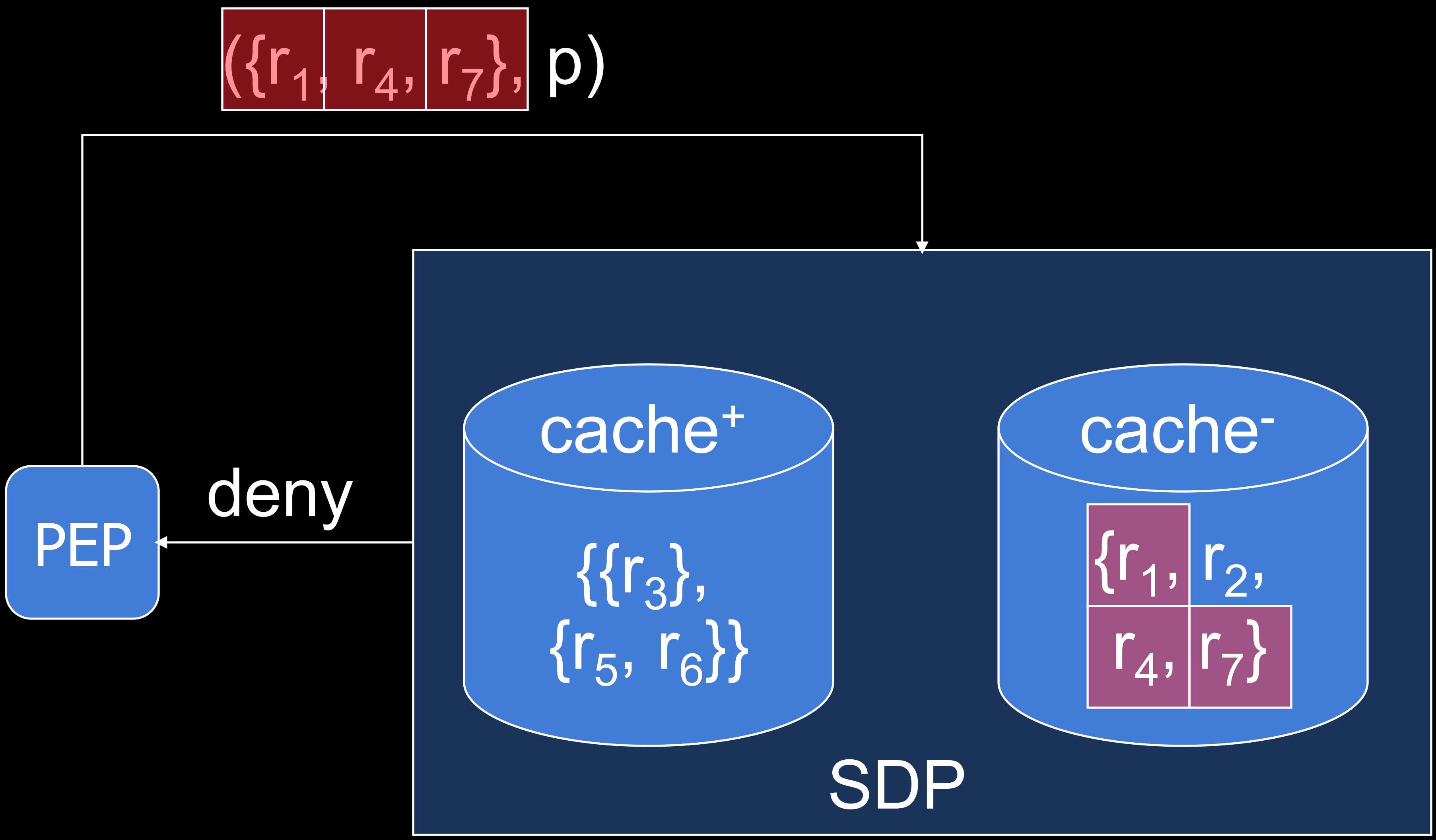
example: SDP receives 2nd deny response



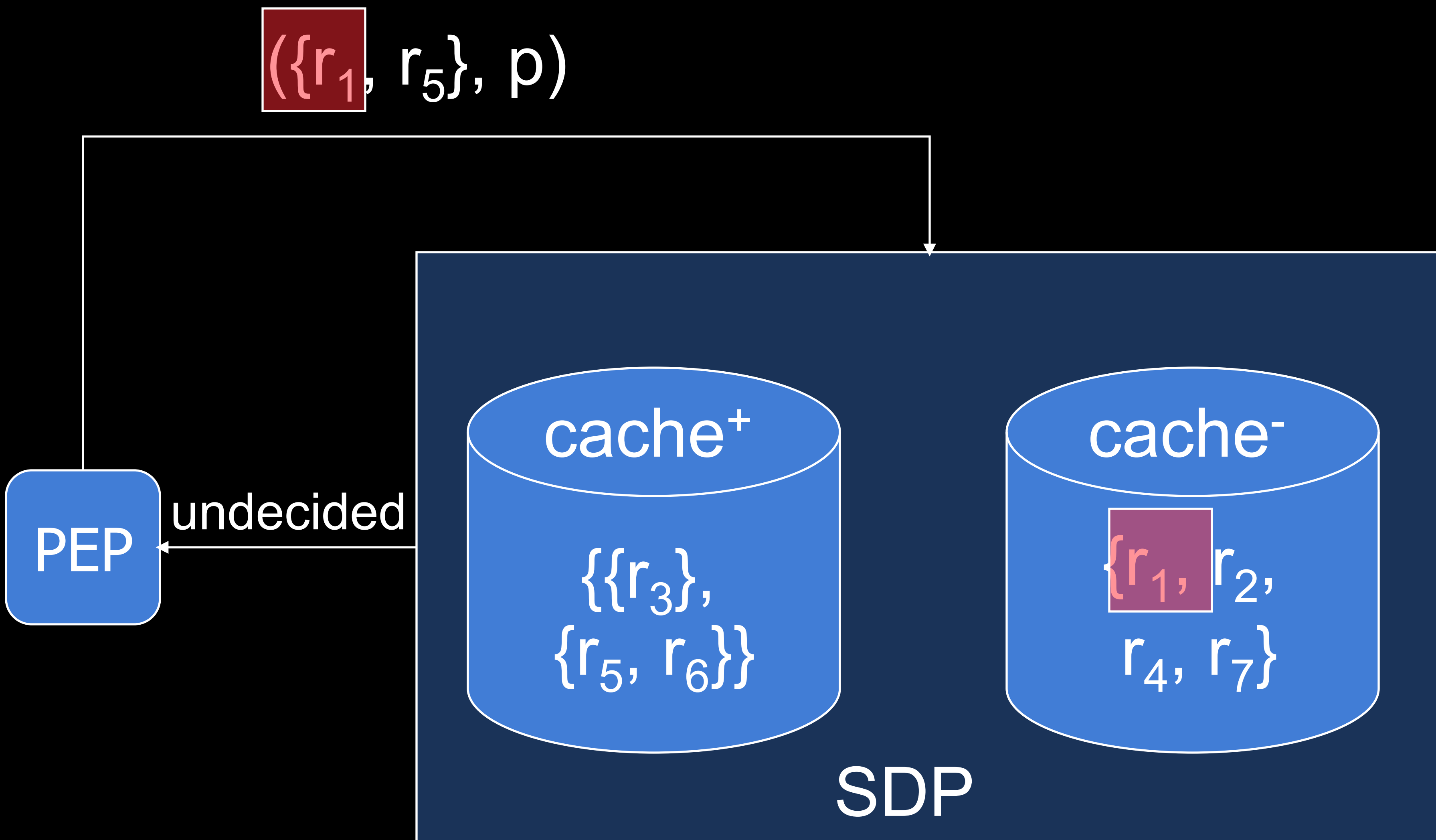
example: SDP makes an allow decision



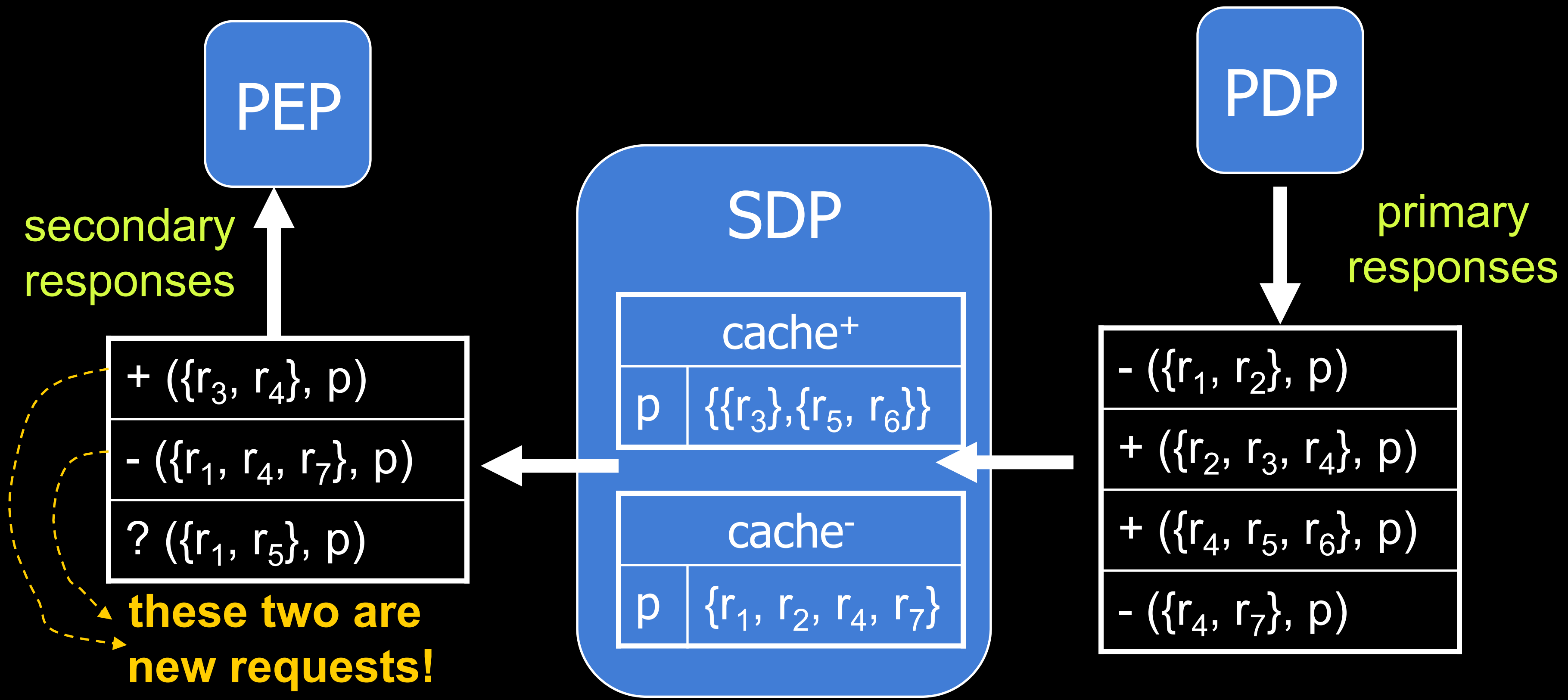
example: SDP makes a deny decision



example: SDP makes an undecided decision



example: summary



- algorithm correctness is proved
  - if the SDP makes any allow or deny decision, the PDP will always make the same decision

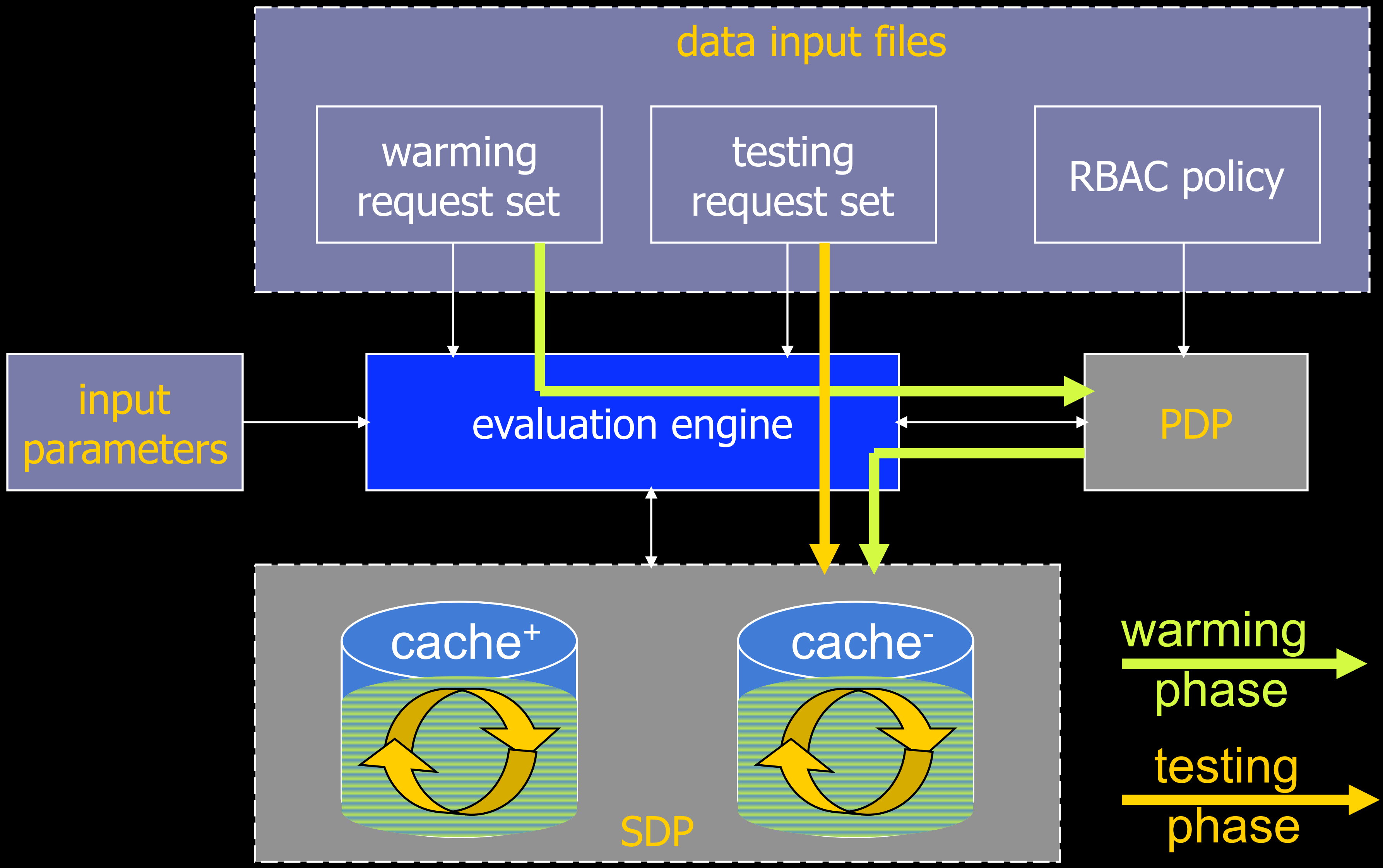
outline

- the overview
- recycling algorithms
- experimental evaluations
- summary and future work

evaluation metrics

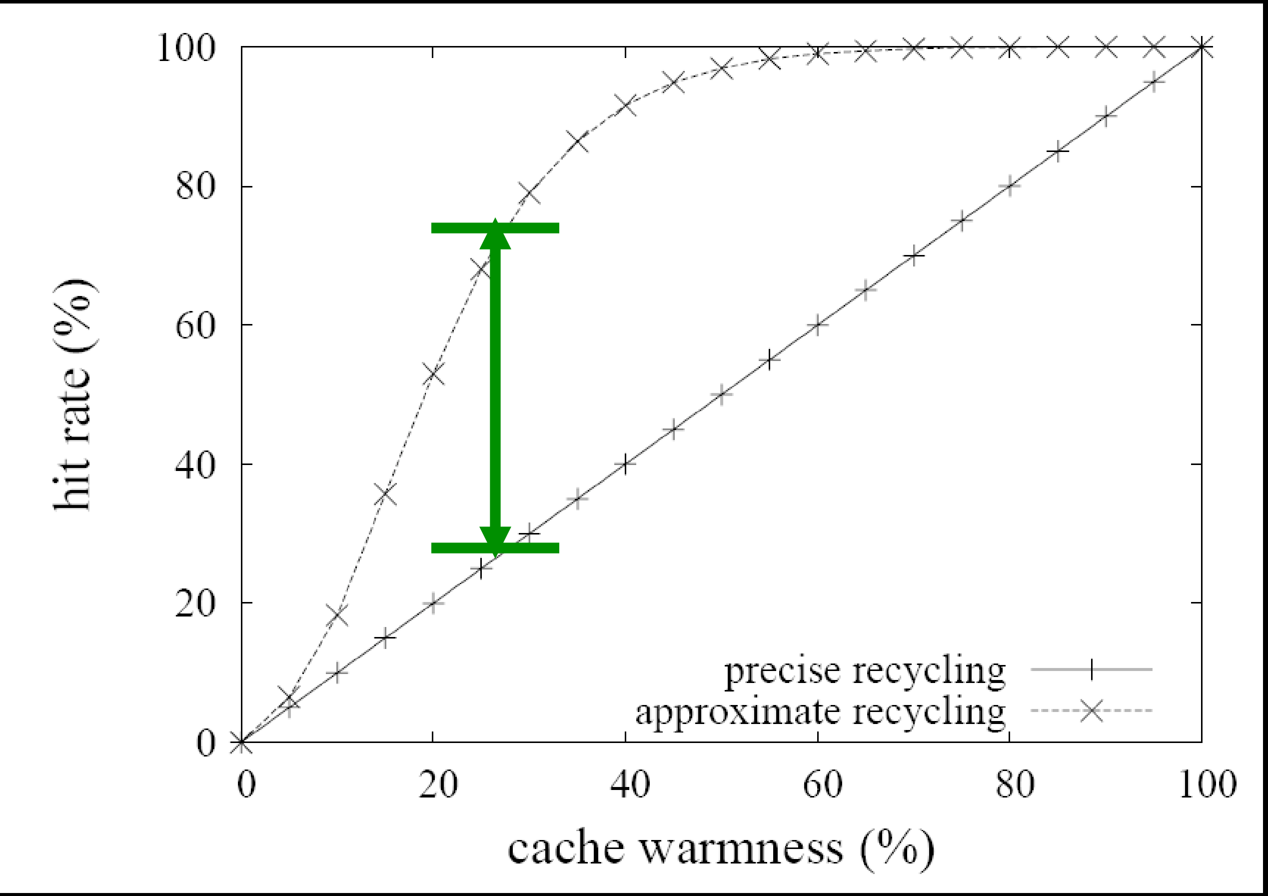
- SDP hit rate
  - a cache hit
    - a request is resolved by the SDP
  - higher hit rate => more requests resolved by the SDP
    - even when the PDP fails => higher availability
    - reducing the load of the PDP => higher scalability
- SDP inference time
    - the time used to infer approximate responses
    - less inference time, more efficient the system

evaluation methodology



hit rate

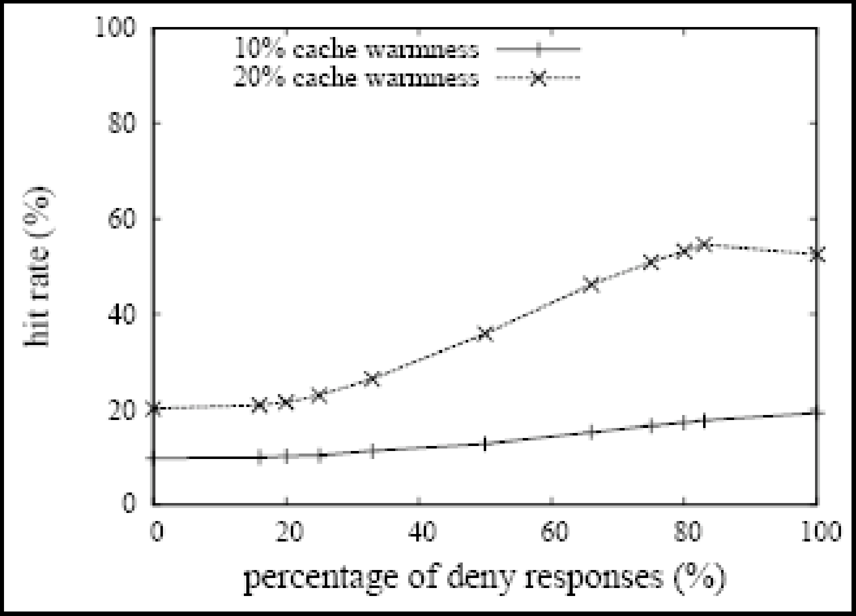
RABC policy: 100 subjects, 1,000 objects, 50 roles

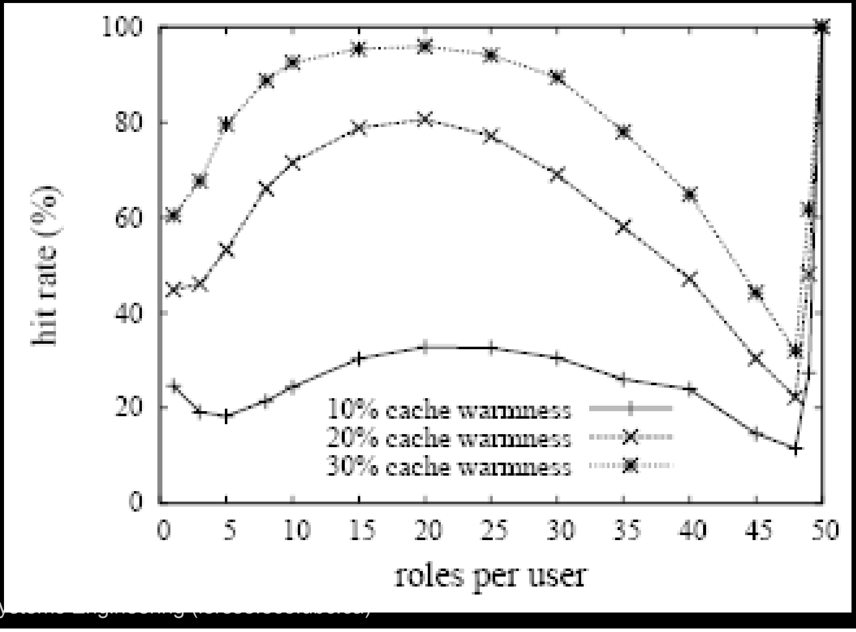


compared with simple caching, hit rate is improved significantly by using $SAAM_{RBAC}$ recycling algorithm

the impact of various system parameters

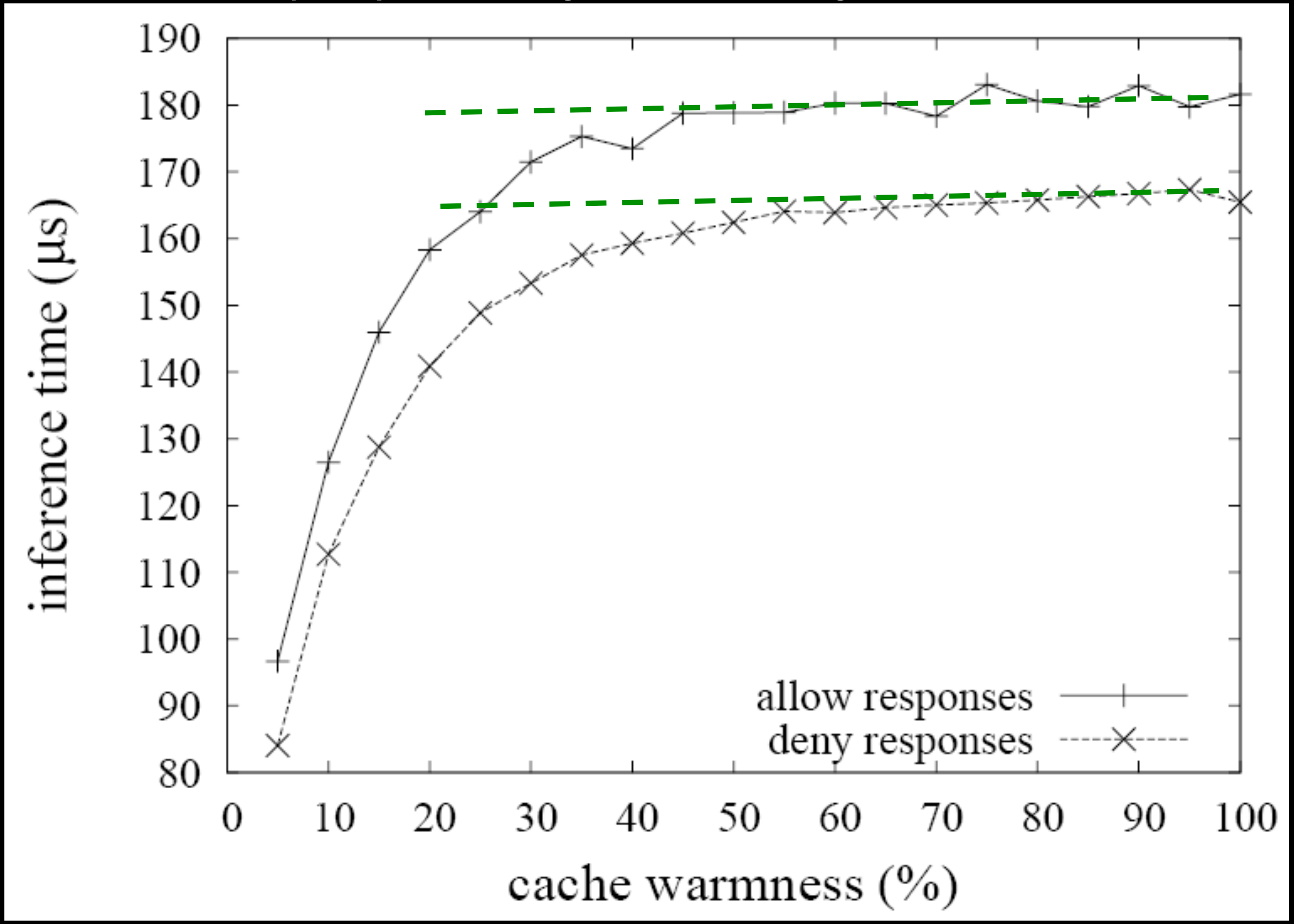
- the percentage of deny responses
- the number of roles
- the number of roles assigned per permission
- the number of roles assigned per user
- the popularity distribution of role assignment
- ...





inference time

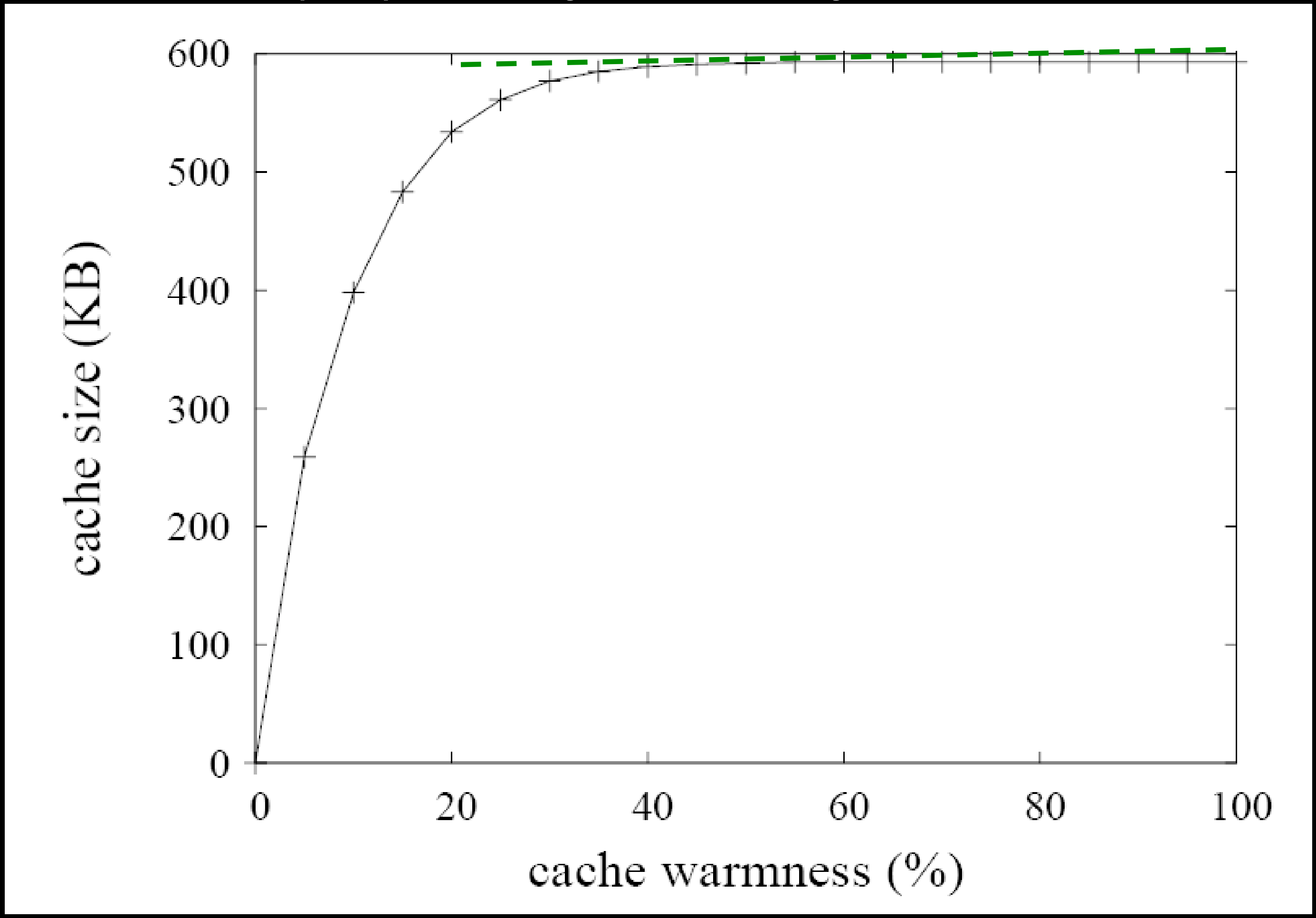
RABC policy: 100 subjects, 1,000 objects, 50 roles



inference time stabilizes

cache size

RABC policy: 100 subjects, 1,000 objects, 50 roles

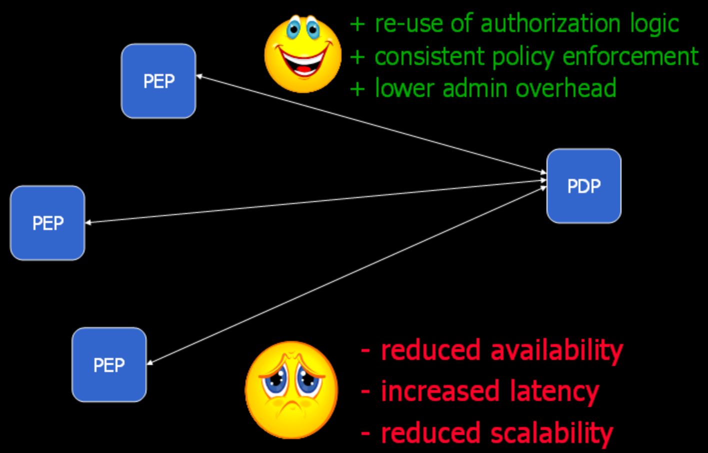


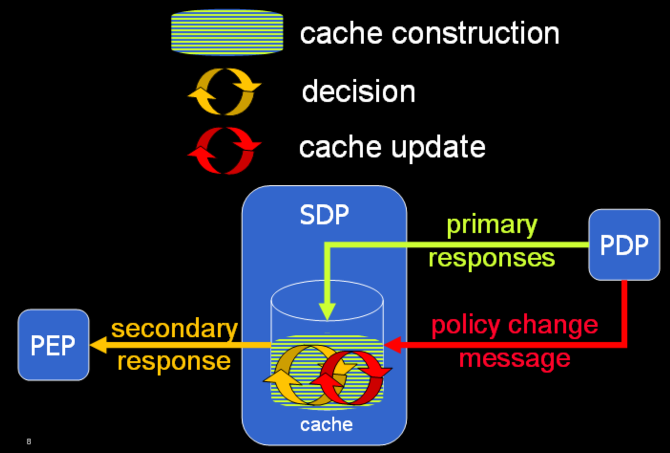
cache size stabilizes

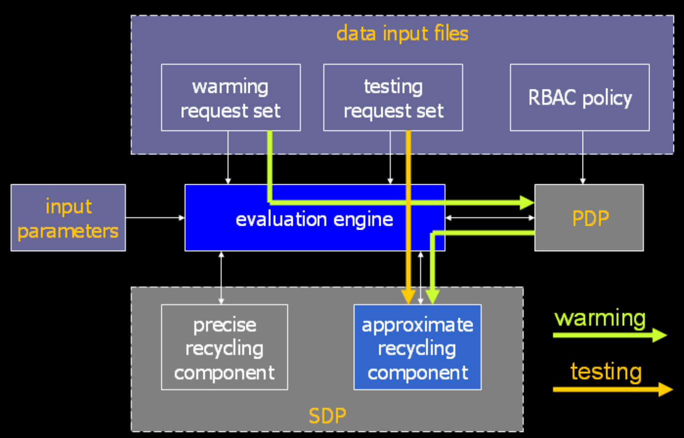
outline

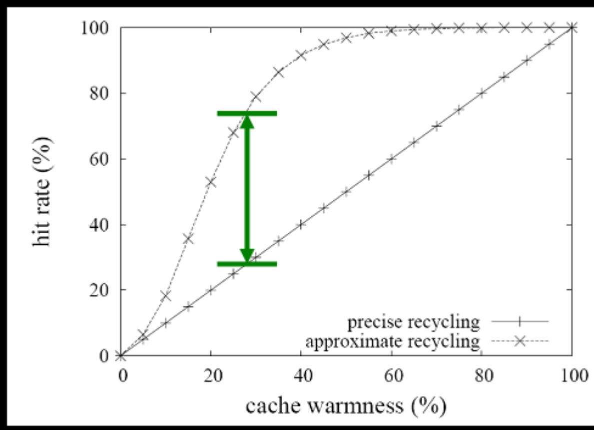
- the overview
- recycling algorithms
- experimental evaluations
- summary and future work

summary


PEP
PEP
PEP
PDP
+ re-use of authorization logic
+ consistent policy enforcement
+ lower admin overhead
- reduced availability
- increased latency
- reduced scalability


cache construction
decision
cache update
SDP
primary responses
PDP
PEP
secondary response
policy change message
cache


data input files
warming request set
testing request set
RBAC policy
input parameters
evaluation engine
PDP
precise recycling component
approximate recycling component
SDP
warming
testing


hit rate (%)
cache warmness (%)
precise recycling
approximate recycling
0
20
40
60
80
100

future work

- when role hierarchy is available
- cache replacement algorithm
- experiment with real enterprise RBAC policies and request traces

We are looking for policies and traces from real applications! If you are willing to share them, please talk to me or contact me at: qiangw@ece.ubc.ca