THE UNIVERSITY OF BRITISH COLUMBIA

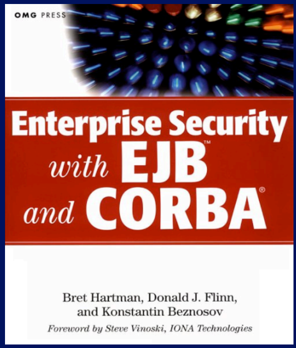
An Overview of The Ongoing Research at LERSSE

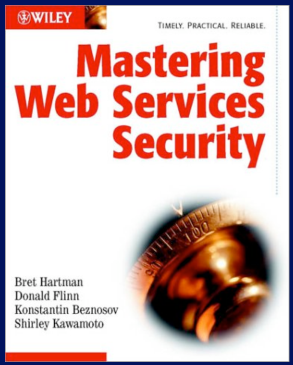
Konstantin Beznosov

http://konstantin.beznosov.net

Who's Konstantin Beznosov

- Education
  - B.S. in Physics (1993), Novosibirsk State University
  - M.S. (1997) & Ph.D. (2000) in CS, Florida Int. Univ.
- Experience
  - US industry (1997-2003): end-user, consulting, and software vendor organizations
  - Assistant Prof., ECE, UBC (2003-present)
- Contributed to
  - OMG
    - CORBA Security revisions
    - Resource Access Decision
    - Security Domain Membership Management
  - OASIS
    - eXtensible Access Control Markup Language v1.0





What's LERSSE?

Laboratory for Education and Research in Secure Systems Engineering

- Research group at the Department of Electrical & Computer Eng. UBC
- People
  - Faculty
    - Konstantin Beznosov, lead (computer security)
    - Sidney Fels (Human Computer Interaction), lead of HCT Lab
  - 2 Ph.D. students
  - 5 Master students + 2 joining in September

http://lersse.ece.ubc.ca

Research Directions and Projects

1. engineering security mechanisms
   - *CORBA Security, RAD,* AAS, **RAD JACCet**, *SDMM, attribute function, EASI, composable authorization engines,* **JAMES**, **AC mech. eval.**
2. access control models & languages
   - CORBA-RBAC, RelBAC *XACML v1.0*, **SAAM**, **probabilistic trust**
3. engineering secure software
   - **agile security assurance**
4. network security
   - **MC-SSL**
5. critical infrastructure interdependencies
   - **CITI interdependencies**
6. usable security
   - **HOT Admin**

Legend: **current**, *back in industry*, presented

outline

- motivation & context: practical security engineering
- engineering secure software
  - agile security assurance
- engineering security mechanisms
  - JAMES
    - SAAM
  - composable authorization engines
- security usability
  - HOT Admin
- network security
  - MC-SSL

THE UNIVERSITY OF BRITISH COLUMBIA

practical security engineering: motivation & context

why aren't secure systems everywhere?

almost completely insecure, or
"secure" but

- too expensive and error-prone to build
- too complex to administer
- inadequate for real-world problems
- forever

what can be done about it?

gradual improvements towards

- inexpensive and error-proof to build
- effective and inexpensive in administration
- adequate for problem domains
- easy and inexpensive to change and integrate

separation of concerns

- application vendors – sell application(s) products
- middleware vendors – sell middleware products
- security vendors – sell security products
- application owners – sell service(s)

THE UNIVERSITY OF BRITISH COLUMBIA

Direction: engineering secure software

Project: agile security assurance

problem

mismatch between

- agile methodologies for software development
- conventional methods for security assurance

hard to assure with agile development

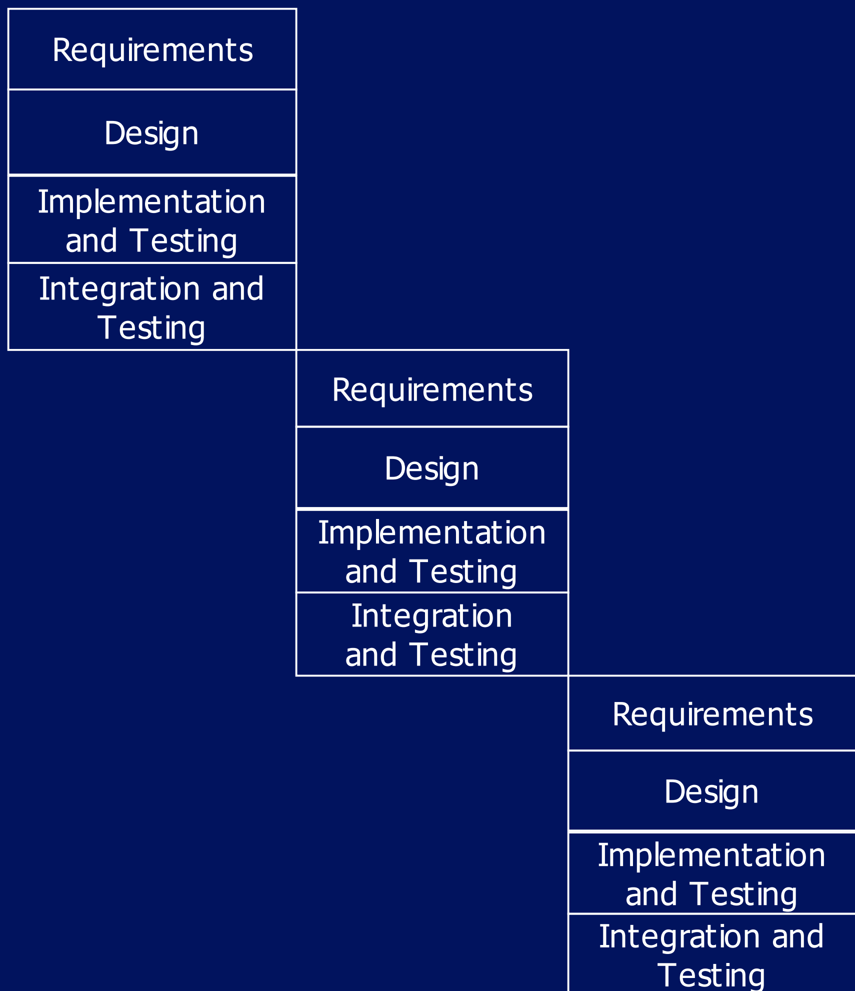
why is addressing the mismatch **important?**

- more security-critical software
- agile methods are here to stay

contribution

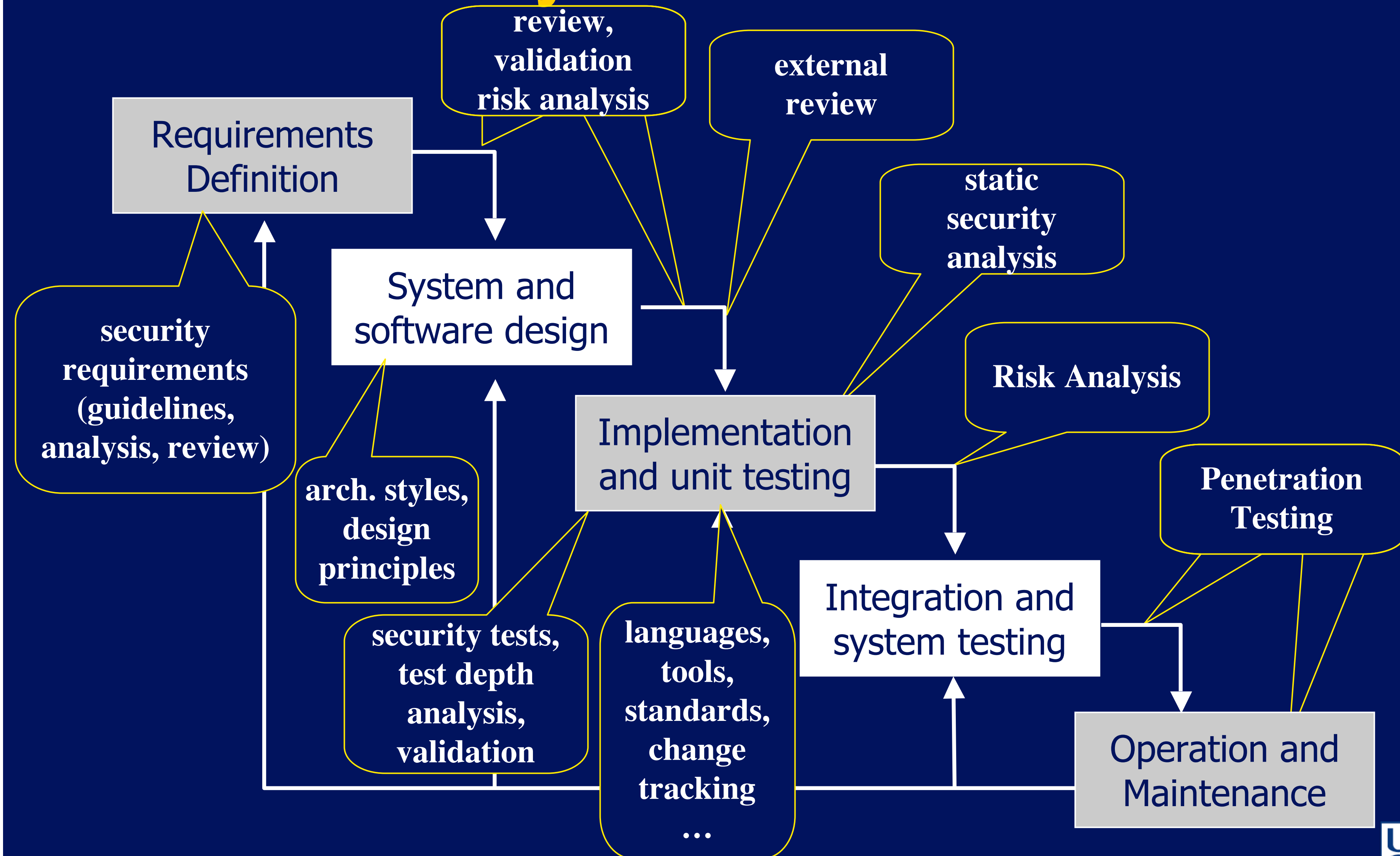
1. examined the mismatch between security assurance and agile methods
2. classified conventional security assurance practices according to the degree of clash
3. suggested ways of alleviating the conflict

what's agile development?



- Characteristics
  - Iterative lifecycle
  - Requirements and design emergence
  - Direct communication
  - Tacit knowledge
- Sample methodologies
  - Crystal
  - Adaptive Development
  - Feature-driven Development
  - Scrum
  - Lean Software Development
  - XP

what's conventional security assurance about?



Adapted from
D. Verdon and G. McGraw, "Risk analysis in software design," *IEEE Security & Privacy*, vol. 2, no. 4, 2004, pp. 79-84.

solution(s)?

If the mountain will not go to Mahomet,
let Mahomet go to the mountain. (proverb)

examination results

Assurance relies on third party

- reviews
- evaluation
- testing

Points of clash

1. direct communication and tacit knowledge
2. iterative lifecycle
3. design refactoring
4. testing “philosophy”

(mis)match classification

1. natural match

   e.g., XP pair programming ♥ internal review & coding standards

2. methodology-neutral

   e.g., language (e.g., Java, C# vs. C, C++),
   version control and change tracking

3. can be (semi-)automated

   e.g., code static analysis,
   security testing/scanning

4. mismatch (≈ 50%)

   e.g., external review, analysis,
   testing, validation change authorization

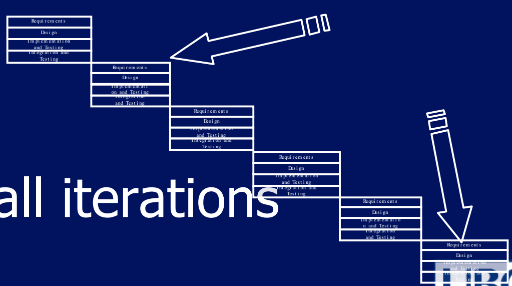
alleviating the mismatch

for (semi)-automatable

- increase acceptance through **tools**
- codify security knowledge in tools
  - automated fault injection, test generation

for mismatching

- search for new agile-friendly assurance methods
  - direct communication and tacit knowledge
  - iterative lifecycle
  - design refactoring
  - testing “philosophy”
- intermittent assurance
  - apply at the first and last iterations
  - use the results to “align” the development
  - have a security engineer (role) involved in all iterations (Wäyrynen et al. 2004)



summary on agile security assurance

problem

mismatch between agile development & security assurance

contributions

1. **examined** (pain points)
2. **classified** assurance methods
3. **alleviated** (tools, knowledge codification, new methods research, intermittent assurance)

Further research

- tool support
- Knowledge classification
- new assurance methods

Direction: engineering security mechanisms

THE UNIVERSITY OF BRITISH COLUMBIA

Project: Junk Authorizations for Massive-scale Enterprise Services (JAMES)

context

- processor time virtually free
- human time/attention expensive
- commodity computing most cost-effective

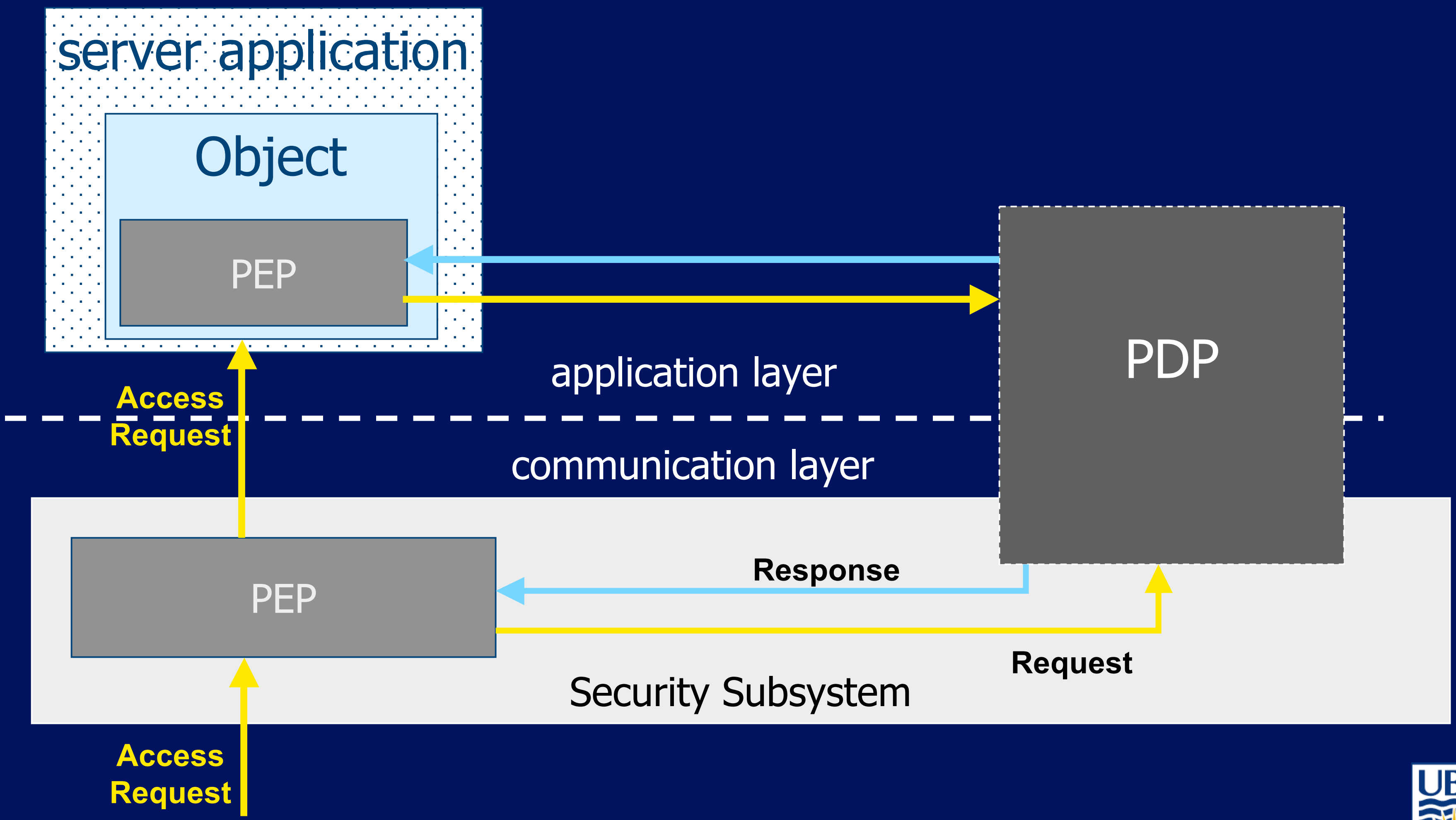
target environments

target environments

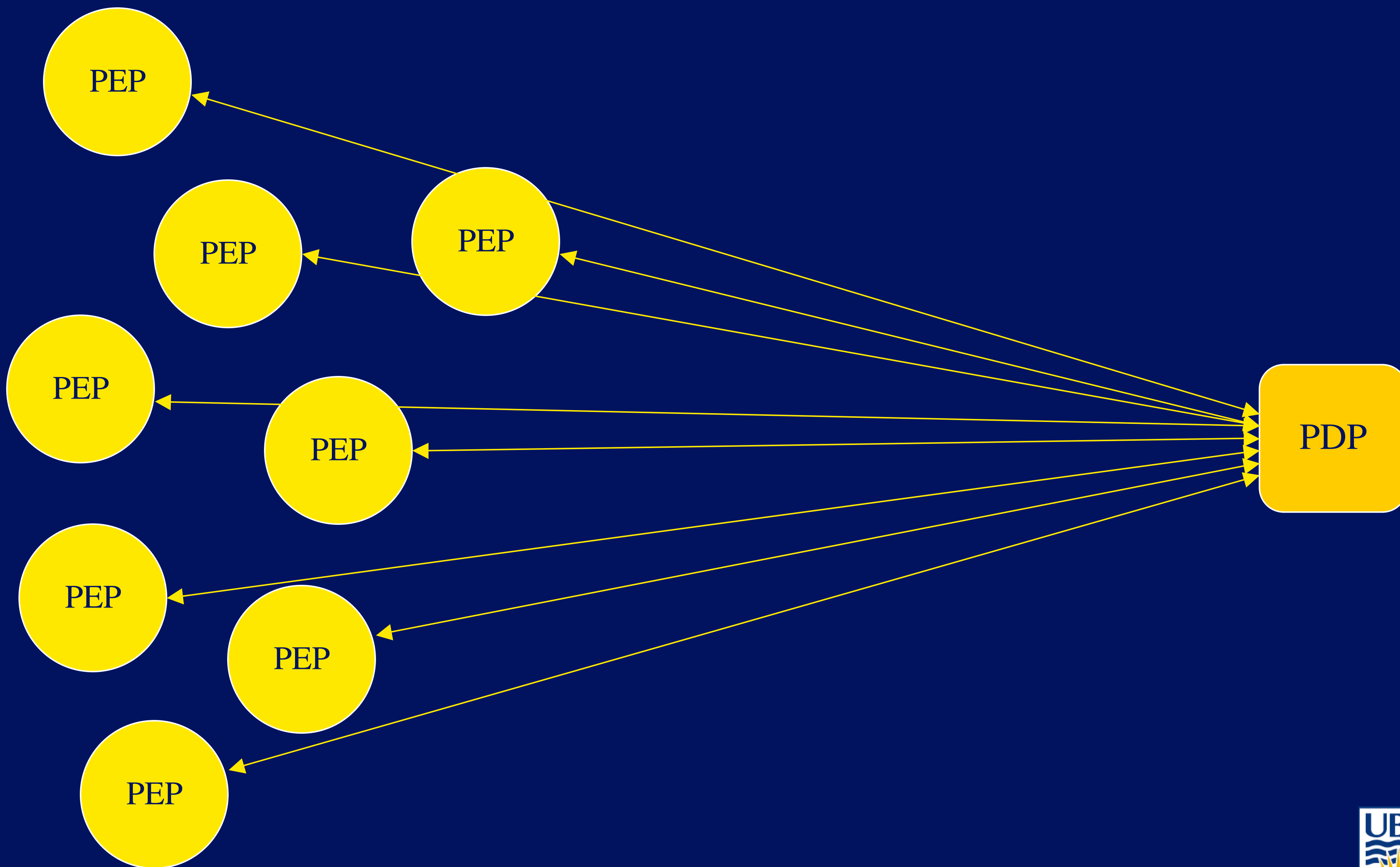
with 0.5M of commodity computing systems

- 0.5--1.5M application instances
- with MTTF of 1 year
  - 1,300--4,000 fail every day
- with availability of 99.9%
  - 500--1,500 unavailable at any given moment

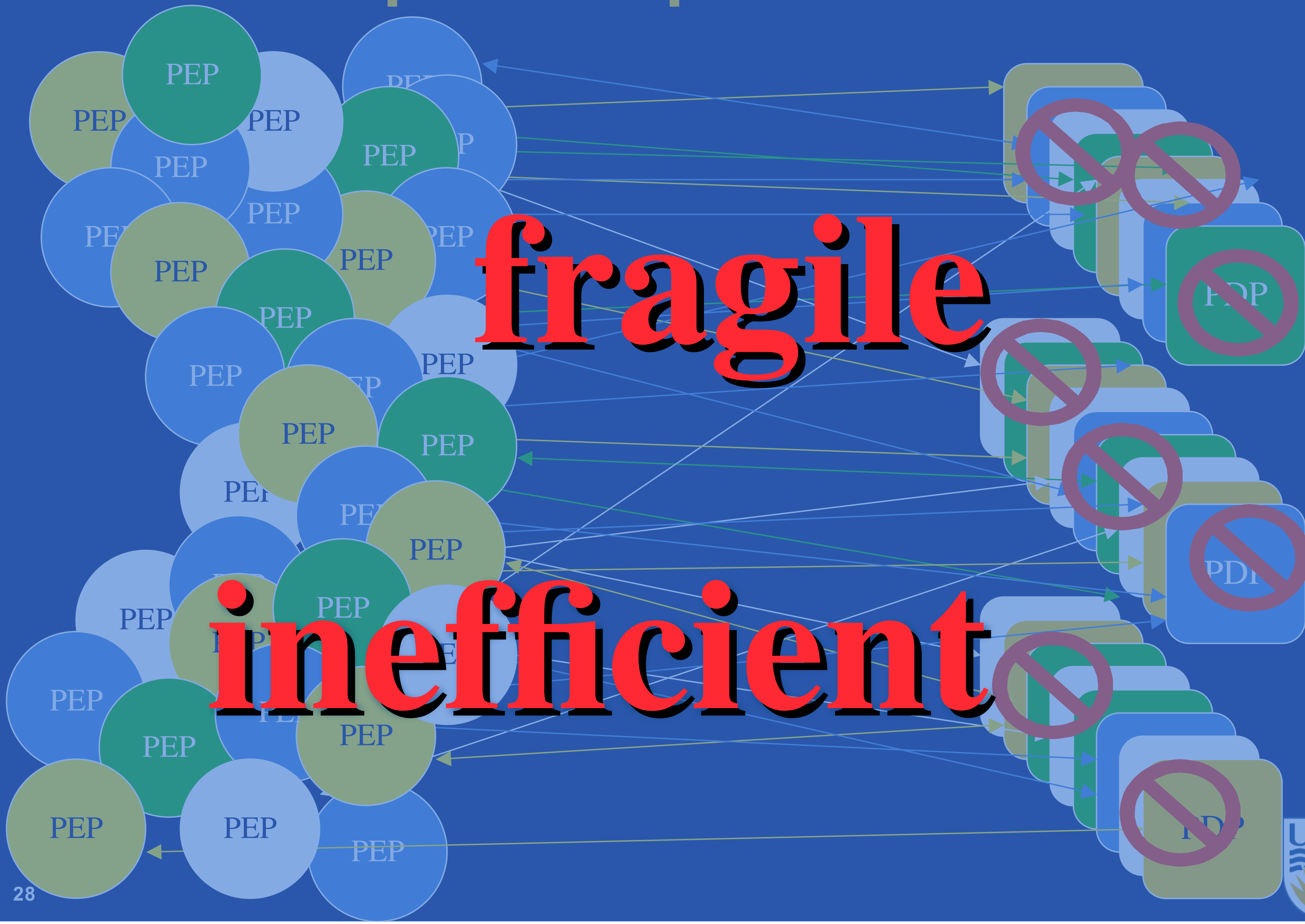
request-response paradigm



enables PDP reuse



results in point-to-point architectures



the new challenge

point-to-point authorization architectures at massive scale

- become too fragile, requiring costly human attention, and
- fail to reduce latency by exploiting the virtually free CPU resources and high network bandwidth

the approach

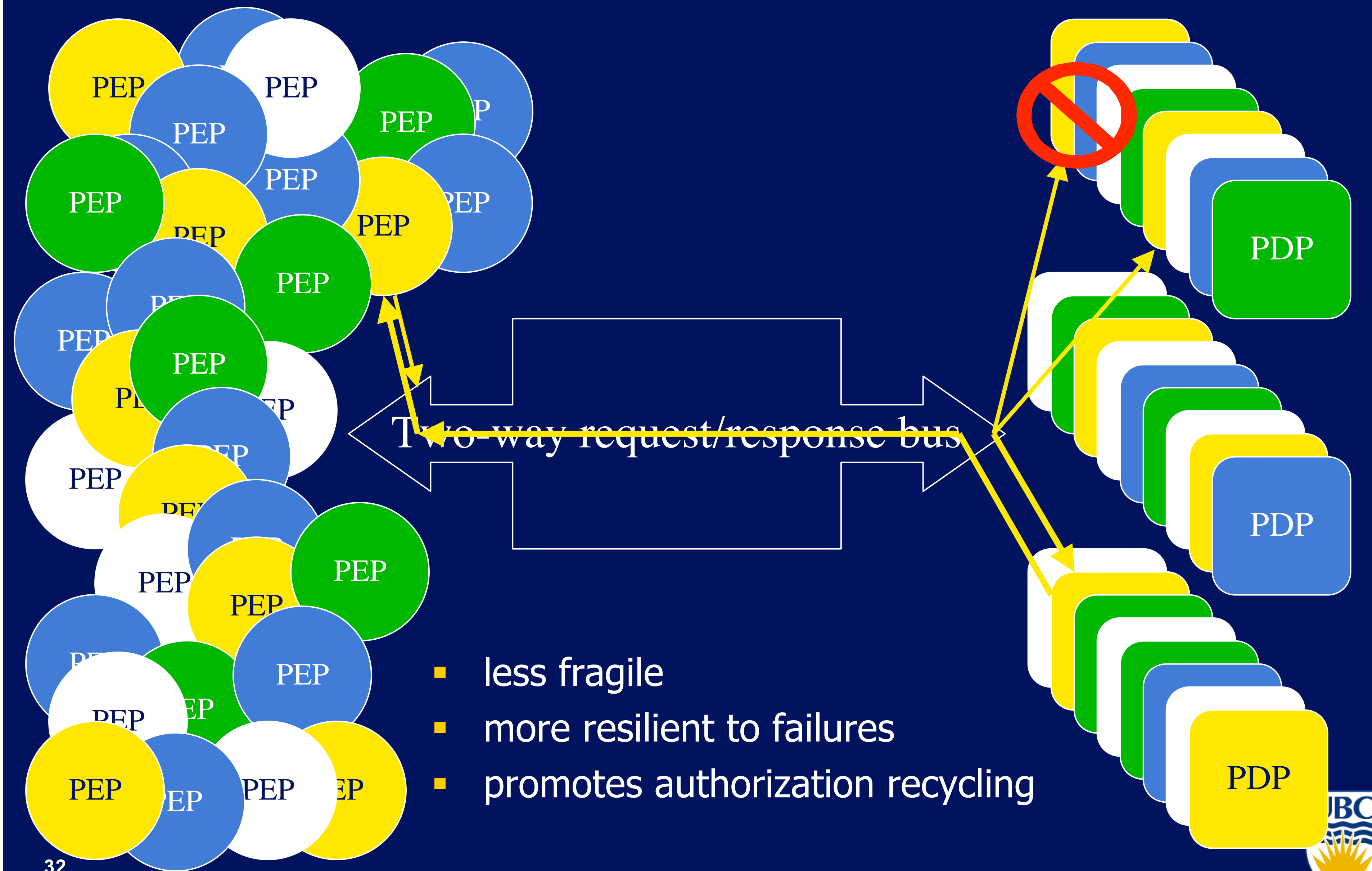
addressing the problem

1. decouple PEP from PDP with publish-subscribe architecture(s)

2. recycle policy decisions

3. flooding

publish-subscribe for policy decisions

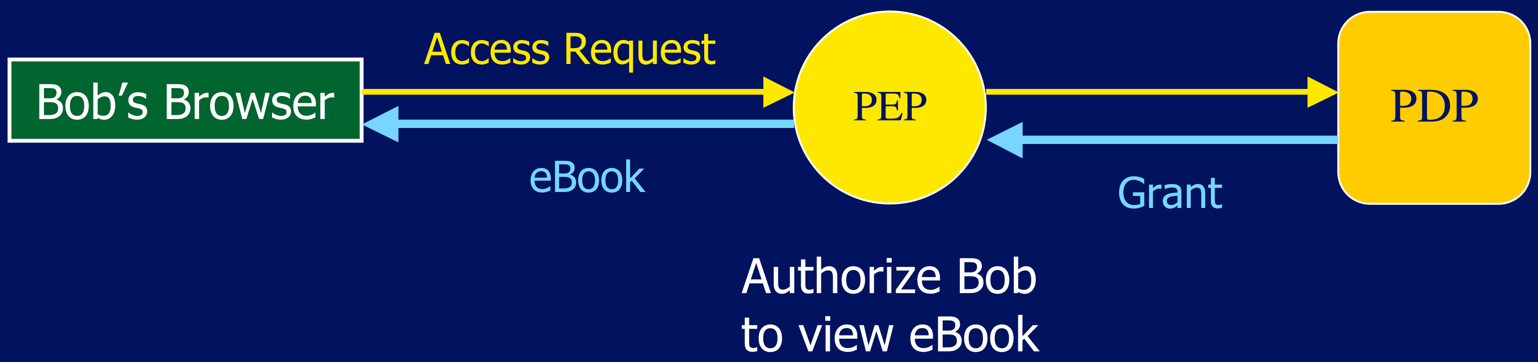


- less fragile
- more resilient to failures
- promotes authorization recycling

recycling authorizations

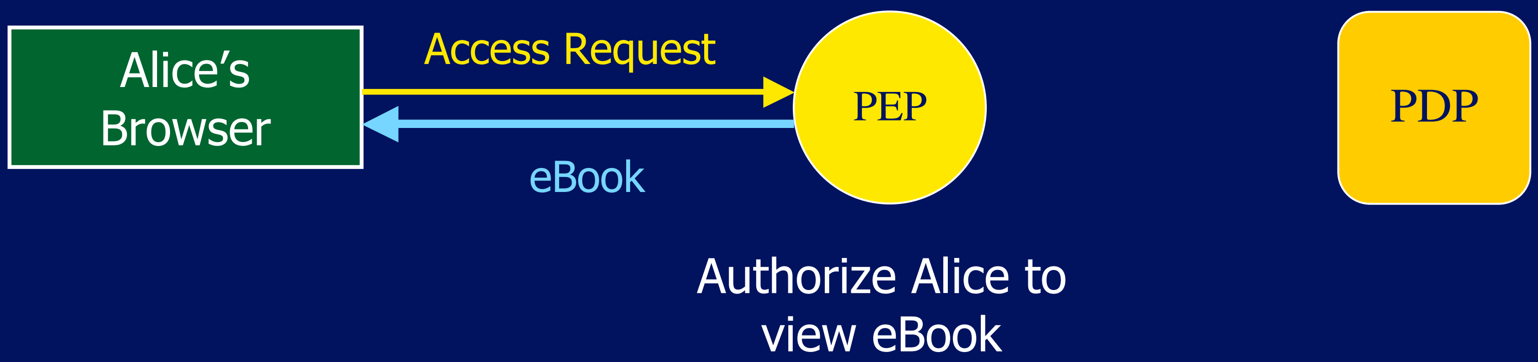
Bob is a *customer*

- He gets authorization to view “Software Design”



recycling authorization

- Alice is a *preferred customer*
  - Has more privileges than Bob
  - System recycles the authorization for Bob and allows Alice to view the book



THE UNIVERSITY OF BRITISH COLUMBIA

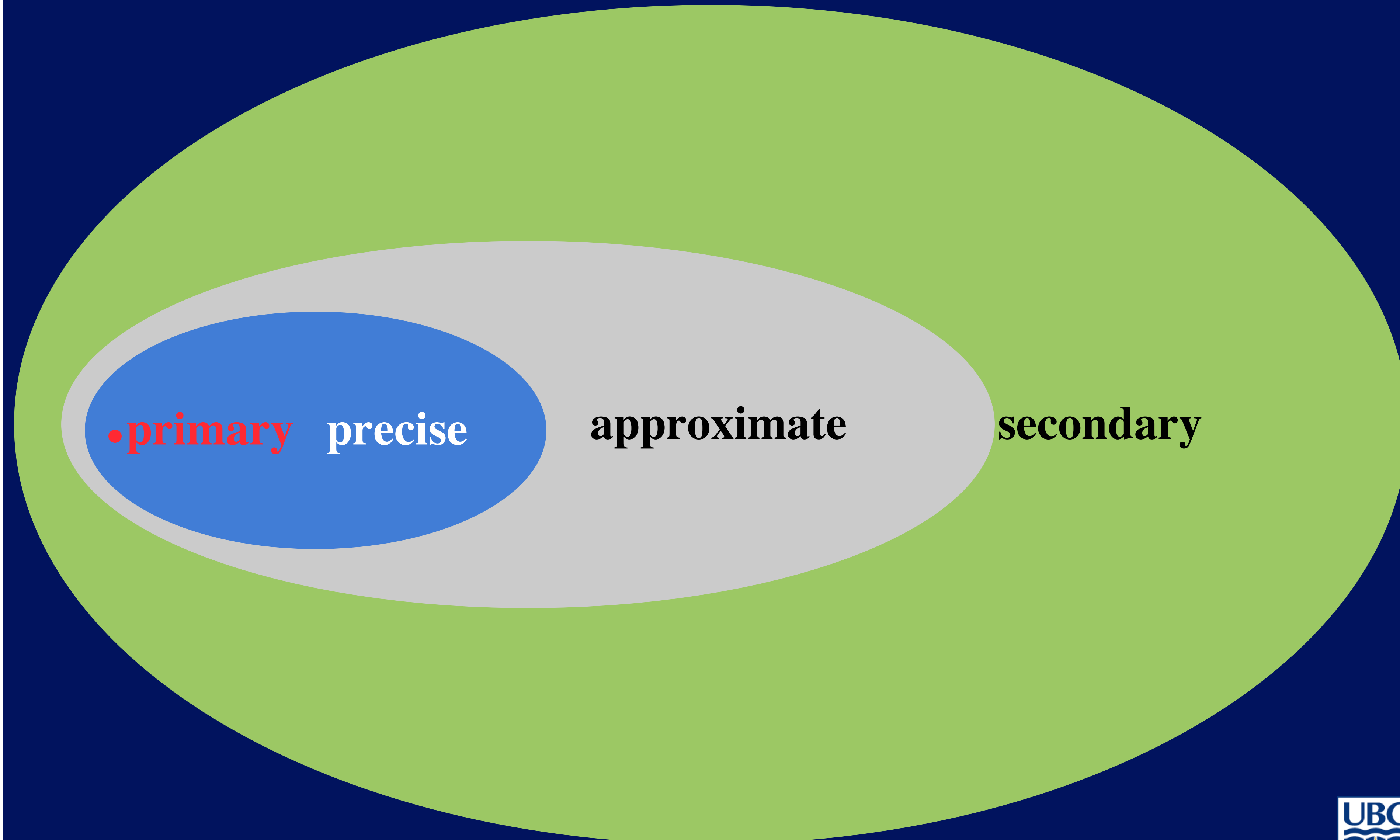
Secondary and Approximate Authorizations Model (SAAM)

basic elements

- request r = <s, o, p, e, i>
  - s -- subject
  - o -- object
  - p -- permission
  - e -- environment
  - i -- request identity

    < s , o, p , e , i >
    <“Bob”, “eBook-123”, “view”, “time=11:30”, “61171092998292”>
- authorization a = <r, d>
  - r -- request
  - d -- decision

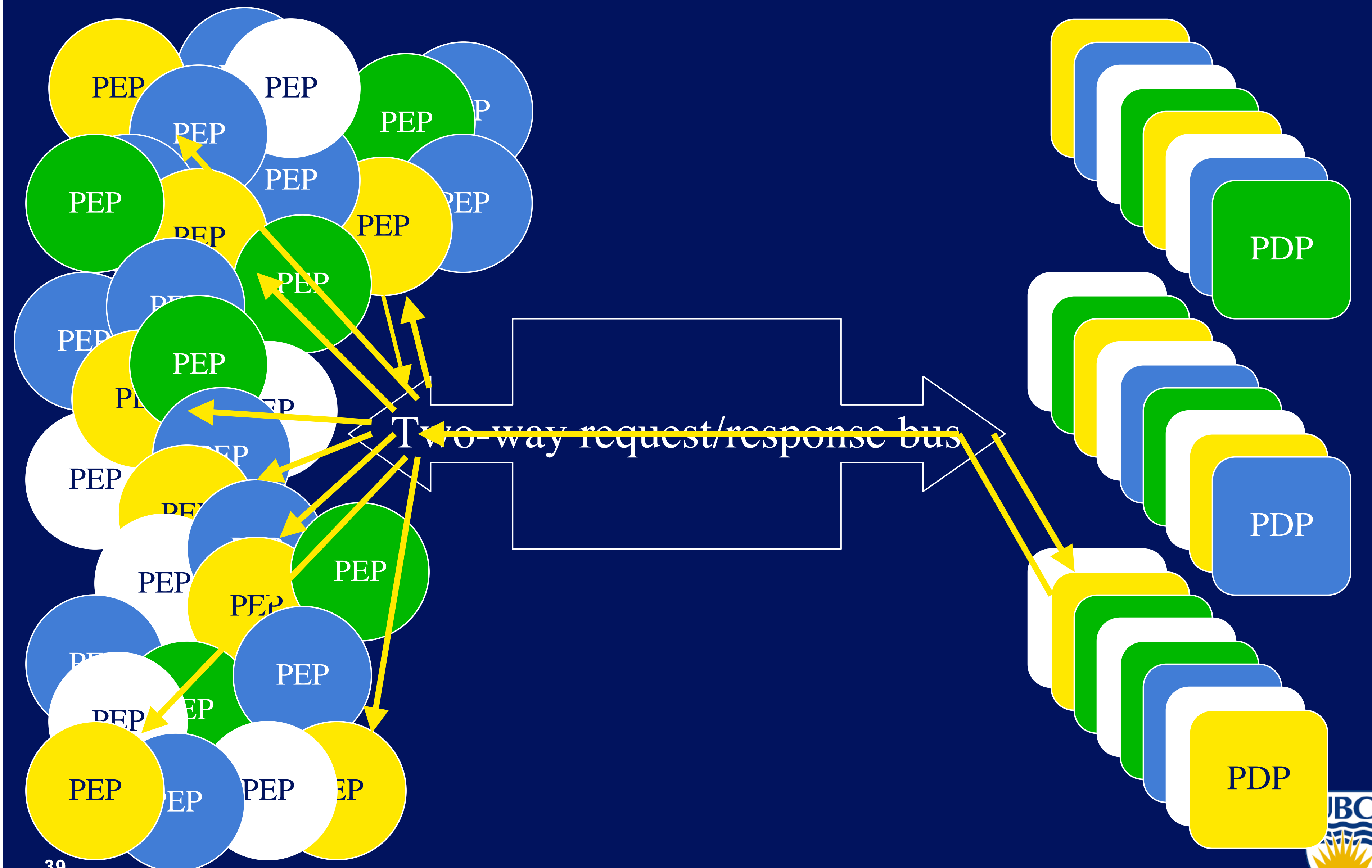
authorization types in SAAM



recycling authorizations

- secondary authorizations
  - re-using decisions made for other, but equivalent, requests
  - example $<s_1, o_1, p_1, e_1, i_1>$ $<s_1, o_1, p_1, e_1, i_2>$
- approximate authorizations
  - re-using decisions made for other, but similar, requests
  - examples
    - $<s_1, o, p, e, i_1>$ $<s_2, o, p, e, i_2>$ $s_1 \geq s_2$
    - $<s, o_1, p, e, i_1>$ $<s, o_2, p, e, i_2>$ $o_1 \leq o_2$
    - $<s, o, p_1, e, i_1>$ $<s, o, p_2, e, i_2>$ $p_1 \leq p_2$

flooding with speculative authorizations



summary for JAMES & SAAM

- problem
  - context and assumptions
    - human time/attention is too expensive
    - CPU resources are virtually free
    - commodity computing is most cost effective
  - target environments
    - massive-scale enterprises with 10^5 machines
  - limitations of point-to-point architectures
    - too fragile, high latency, too expensive to maintain
- approach to address
  - decouple PEP and PDP with publish-subscribe
  - authorization recycling
    - secondary and approximate authorization model (SAAM)
  - flooding

THE UNIVERSITY OF BRITISH COLUMBIA

Project:
composable authorization engines

problem motivation

Mechanism ⟷ Policy

(Enforcement function) | (Decision function a.k.a. Policy engine)

Distributed app. developers/admins have limited choices:

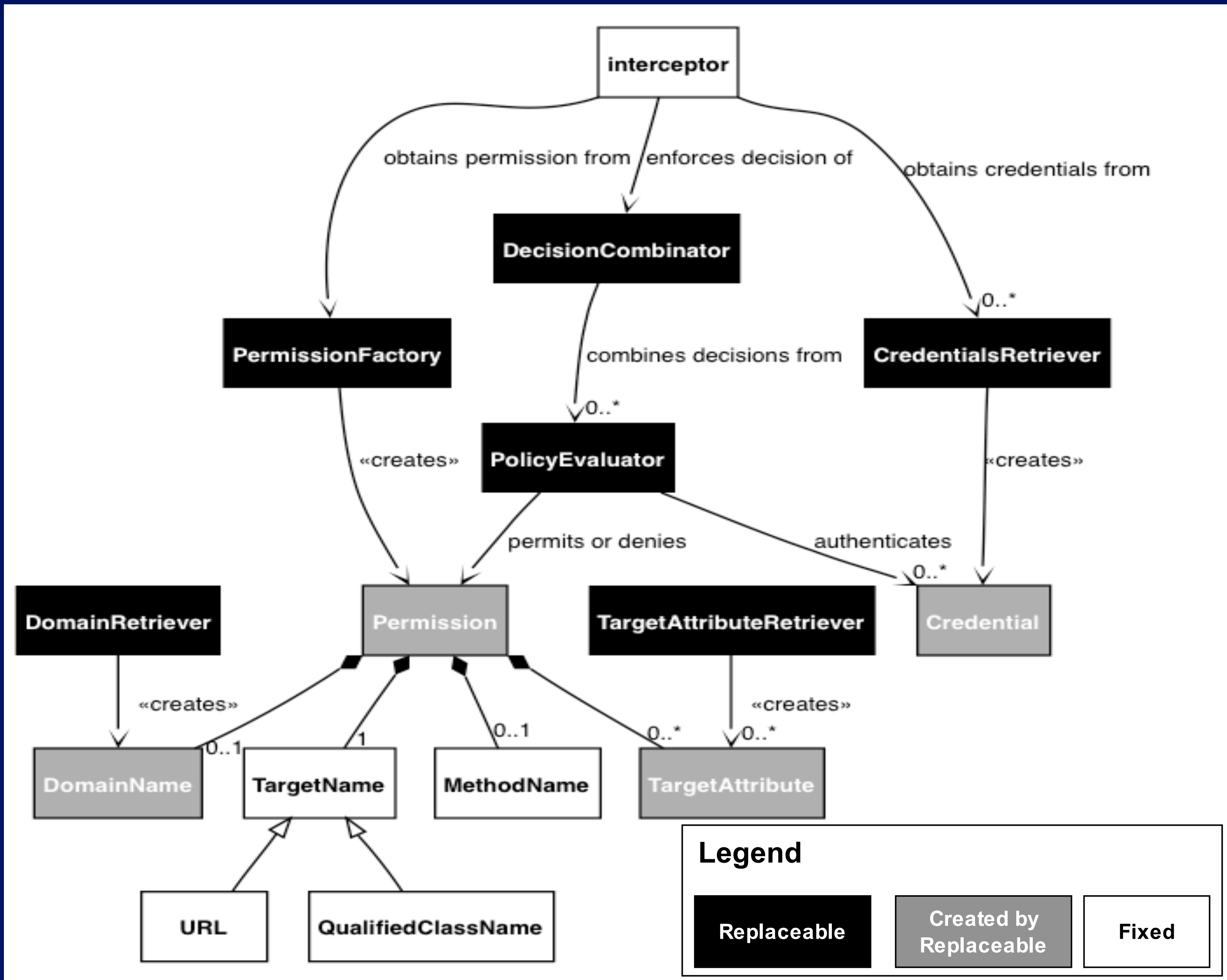
1. Pre-built policy engines with limited capabilities
   - e.g., JAAS default policy file, COM+, EJB authorization
   - Limited support for non-trivial or application-specific policies
2. Pre-built policy engines "one size fits all" generic
   - e.g., CORBA
   - Unnecessary complex and expensive to use
3. "plug-in" APIs for creating custom "do-it-yourself" engines
   - e.g., CORBA Sec. Replaceable, JACC, SiteMinder and alike
   - Hard to do it right

premise

- common policy elements
  - e.g., authorizations based on roles, groups, location
- differences in

1. the weight and composition
   - e.g., location || (role && group) vs.
     role || (location && group)
2. application-specific factors
   - e.g., relations, certification, license

component framework for A&A policy engine



expected benefits

- wide range of supported policies
- "pay as you go" cost of supporting a policy
  - determined by required policy
    - not by policy engine complexity
  - incremental changes proportional to policy Δ-s
    - addition/removal/re-composition of policy components
    - re-use of existing policy logic by developers/administrators

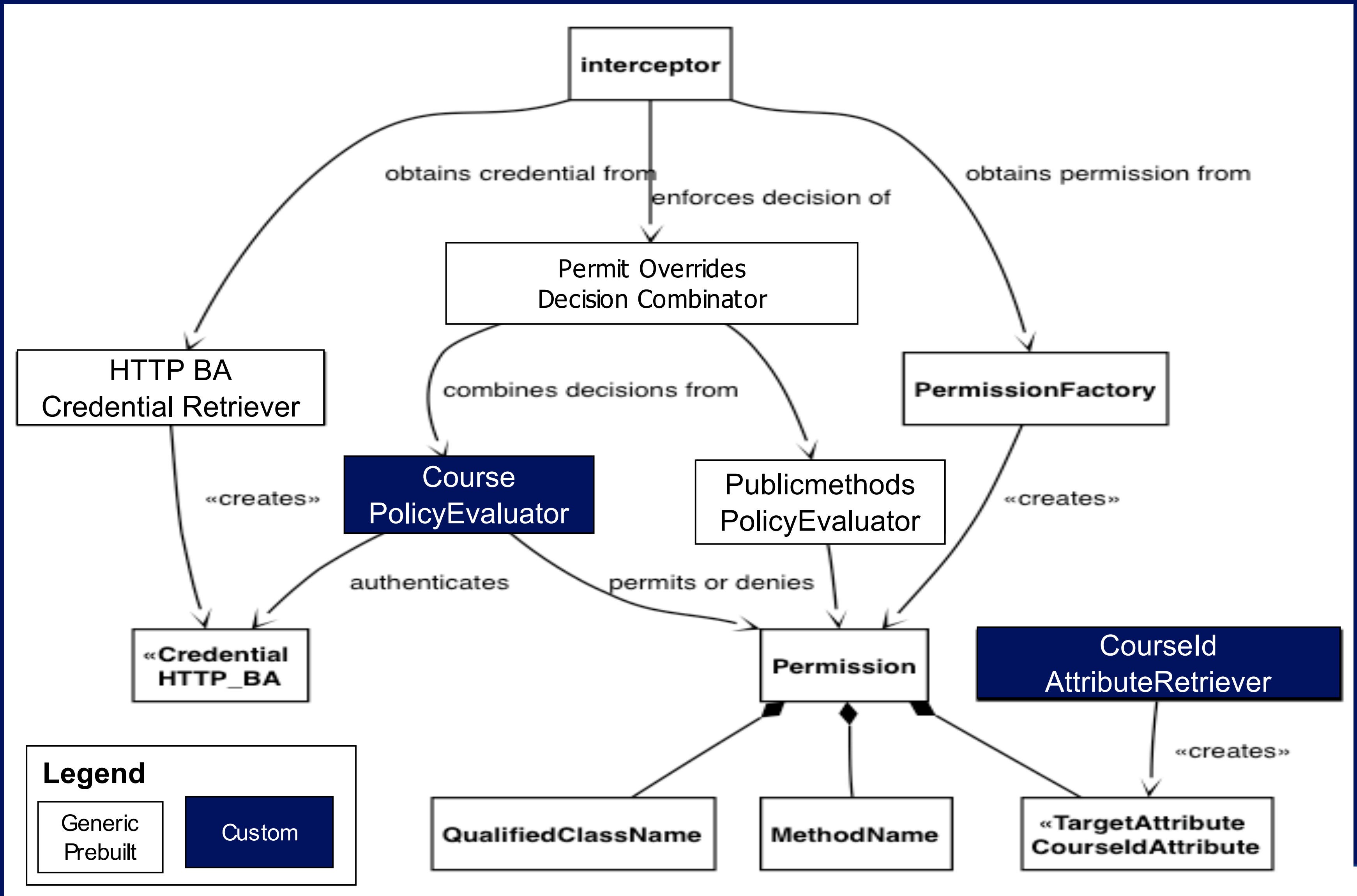
example 1

university course web service

university course web service **policy**

1. Anyone can lookup course descriptions.
2. All users should authenticate using HTTP-BA.
3. Registration clerks can list students registered for the course and (un)register students.
4. The course instructor can list registered students as well as manage course content.
5. Registered for the course students can download assignments and course material, as well as submit assignments.

policy engine assembly for example 1



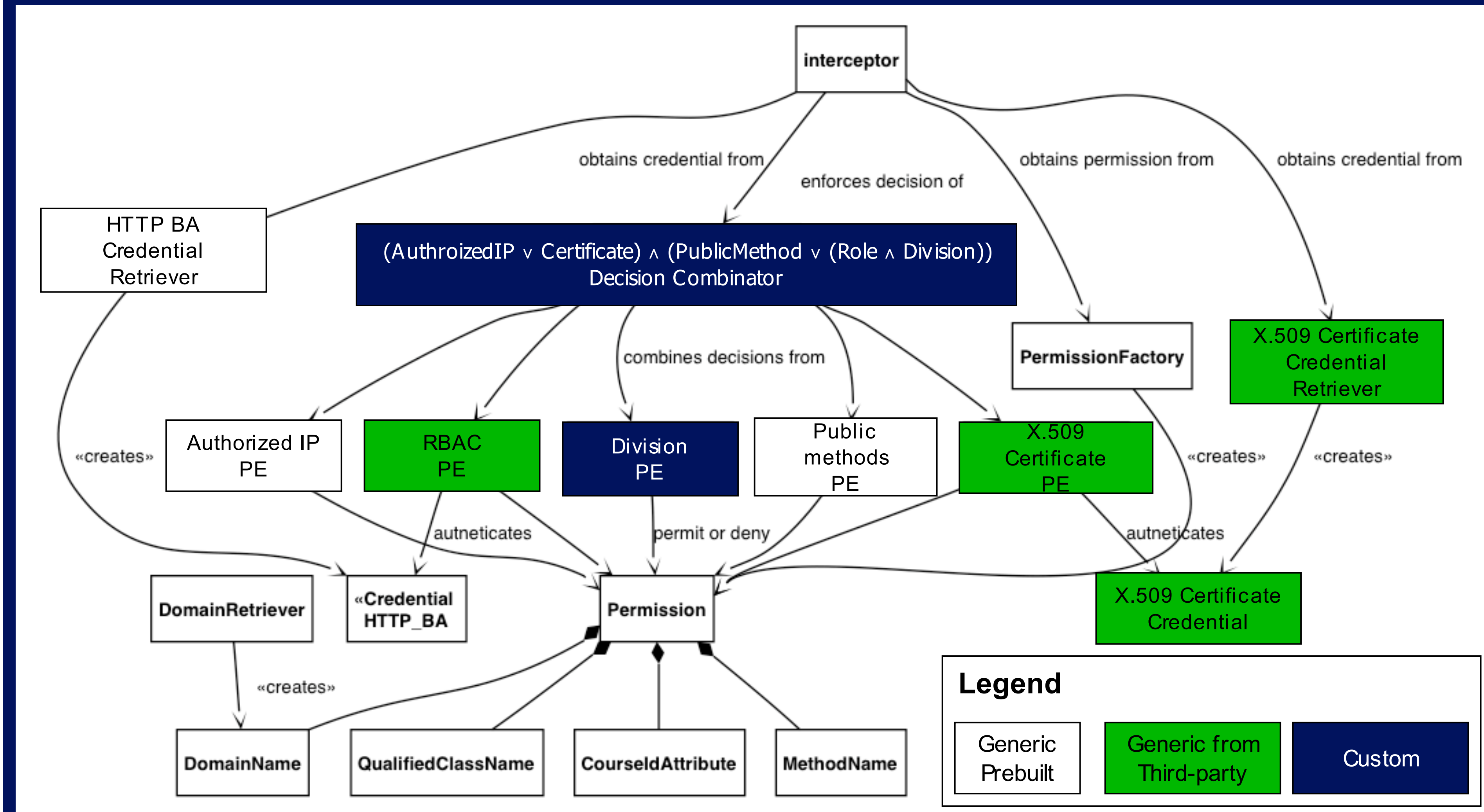
example 2

human resources web service for an international organization

HR web service policy

1. Only users within the company's intranet or those who access the service over SSL and have valid X.509 certificates issued by the company should access.
2. Anybody in the company can look up any employee and get essential information about her/him.
3. HR employees can modify contact information and review salary information of any employee from the same division.
4. HR managers can modify any information about the employees of the same division.

policy engine assembly for example 2



unresolved issues

- validating engine configuration against a given policy
- generating engine configuration for a given policy

Direction: usable security

THE UNIVERSITY OF BRITISH COLUMBIA

Project:
HOT Admin

Human, Organization, and Technology Centred Improvement of IT Security Administration

Konstantin Beznosov, Sidney Fels, Lee Iverson
University of British Columbia
Brian Fisher
Simon Fraser University

overview

- purpose
  1. evaluation methodology for sec. admin. effectiveness
  2. guidelines and techniques to design sec. admin. tools
- problem addressed
  - conflict of human, organizational, and technological forces
- approach
  - resolve the conflict through harmonizing the forces
- work plan (3 years)
  1. pilot studies to fine-tune the methodologies
  2. inventories and an initial analysis through field research
  3. development of models
  4. design of techniques and methodologies
  5. validation and evaluation of the project's key results.
- team
  - Beznosov (security), Fels (interfaces),
    Iverson (collaborations), Fisher (interaction)

purpose

1. methodology for evaluating the effectiveness of the existing IT security administrative tools

2. guidelines and techniques to systematically design effective technological solutions to aid security administrators

problem

classical access control solution

subjects

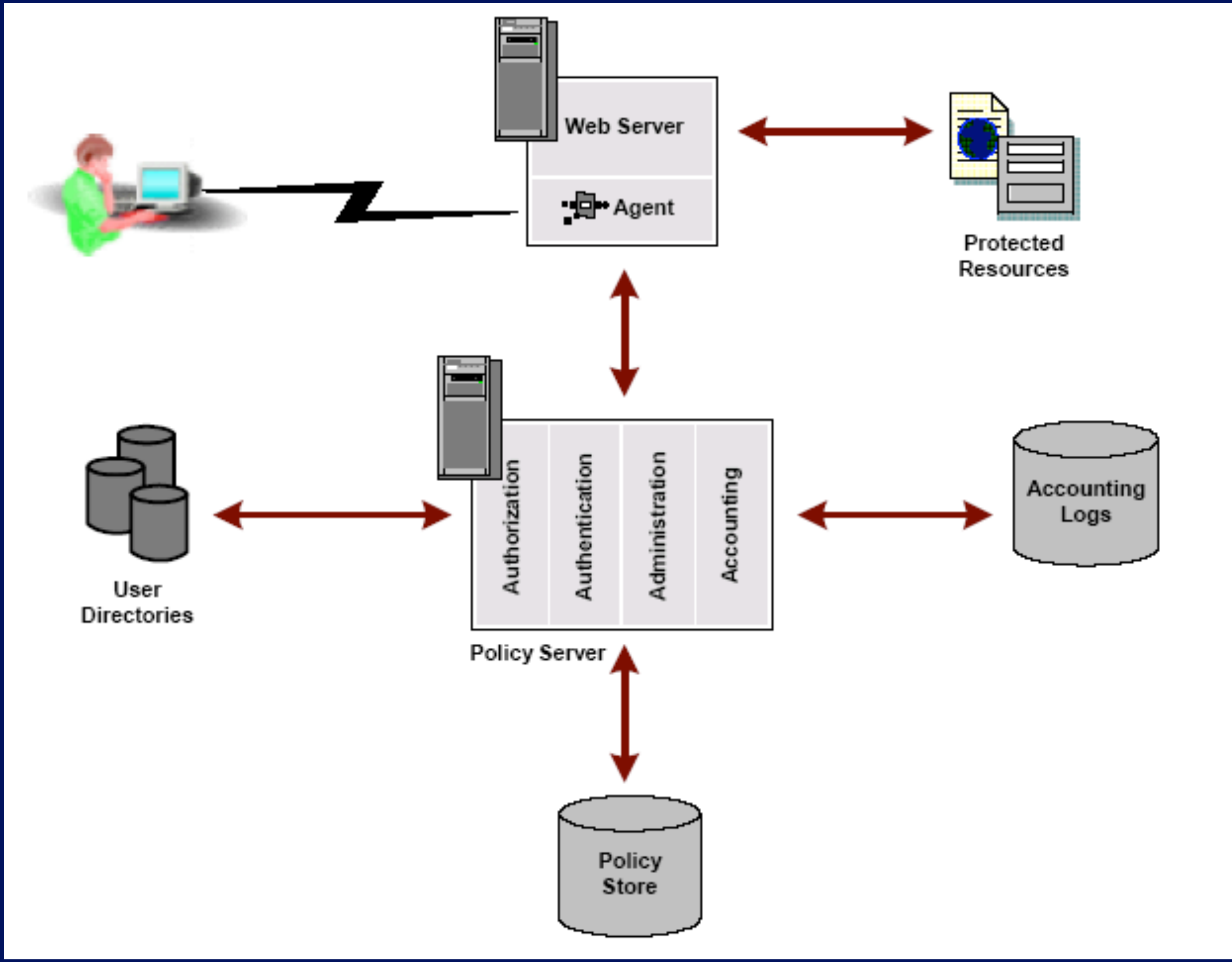
objects

Access Matrix

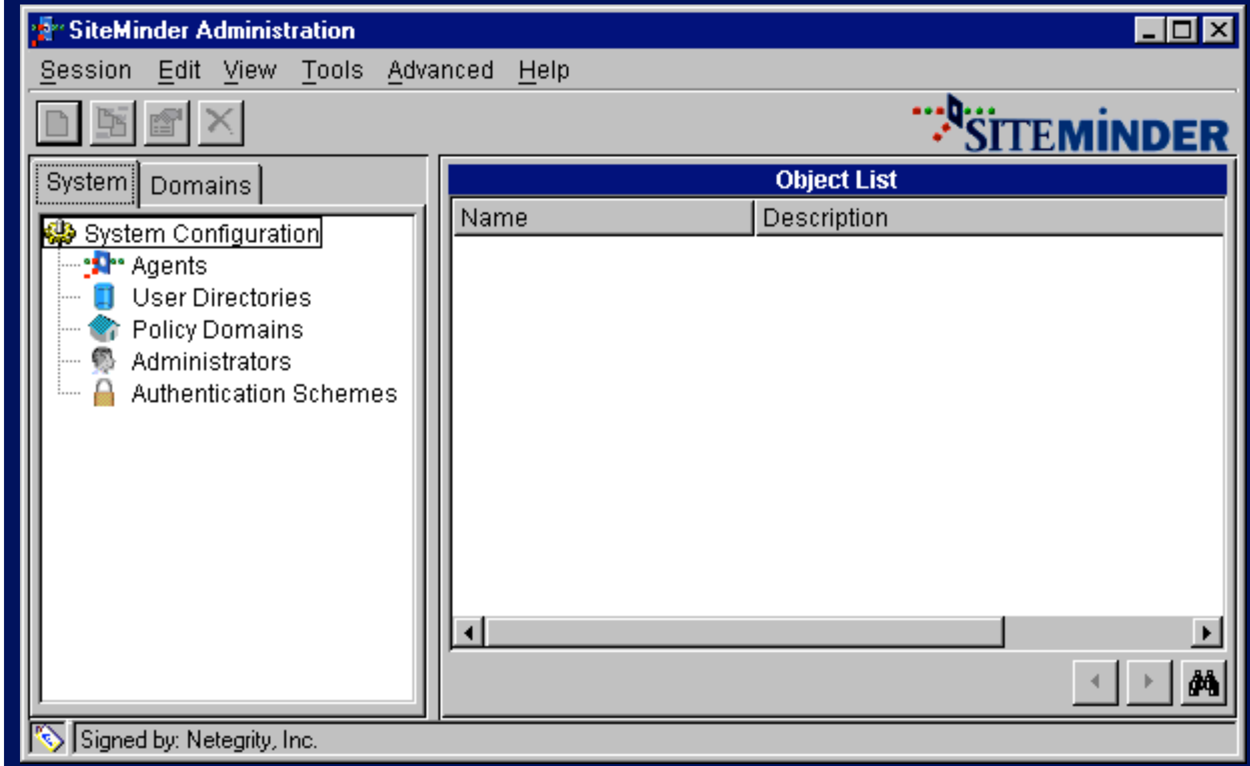
A

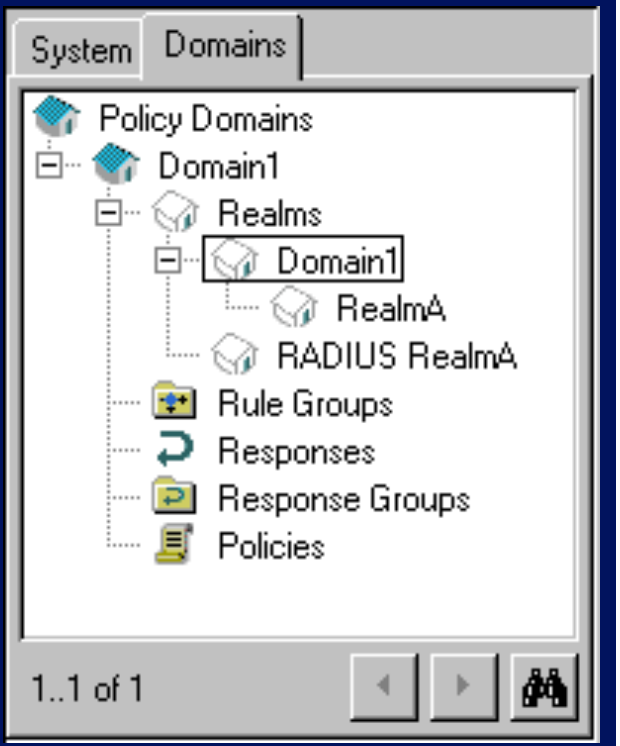
	Domain 1	Domain 2	Domain 3	File 1	File 2	Process 1
Domain 1	*owner control	*owner control	*call	*owner *read *write		
Domain 2			call	*read	write	wakeup
Domain 3			owner control	read	*owner	

enterprise-scale security server

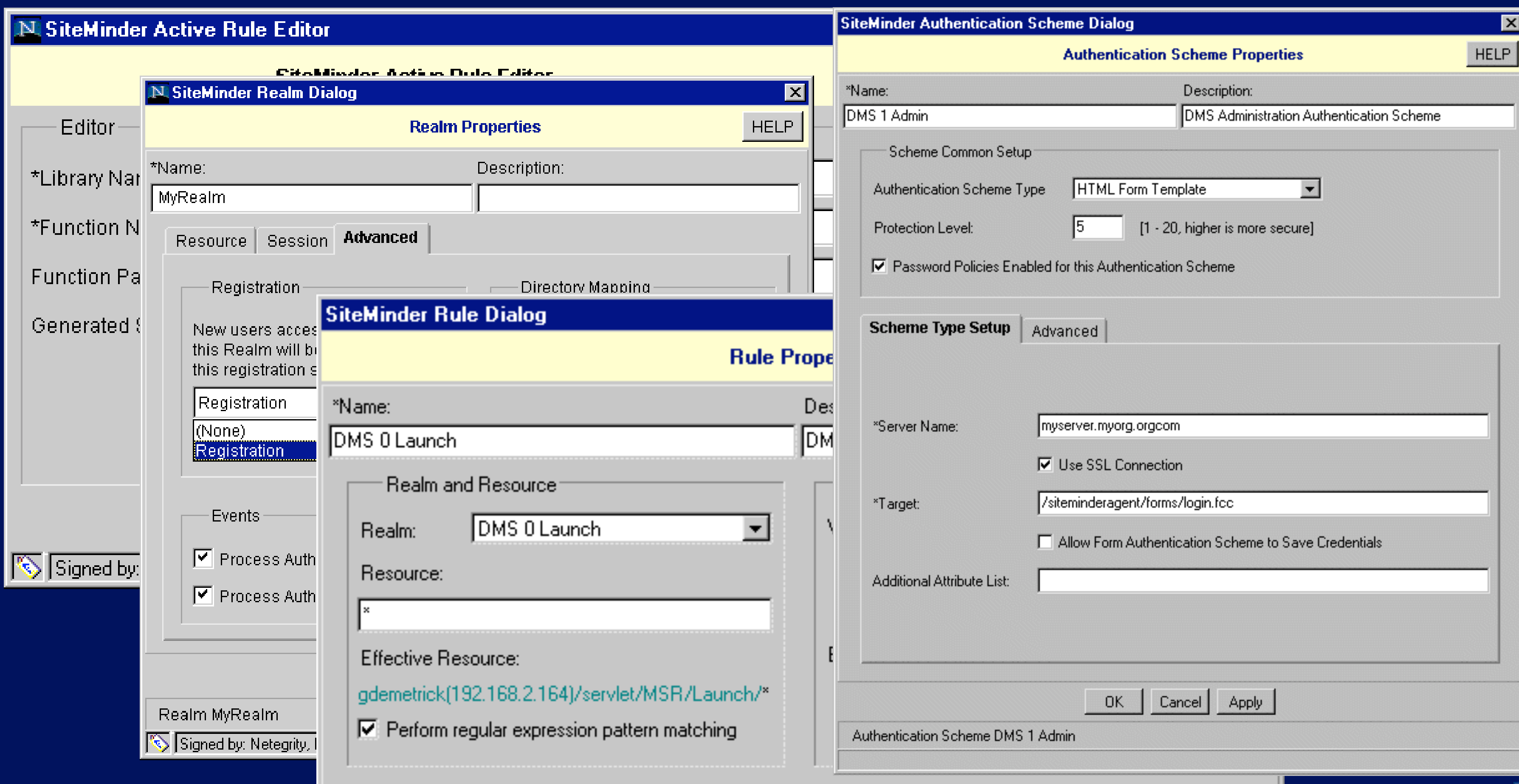


everything starts with simple tree-like structure

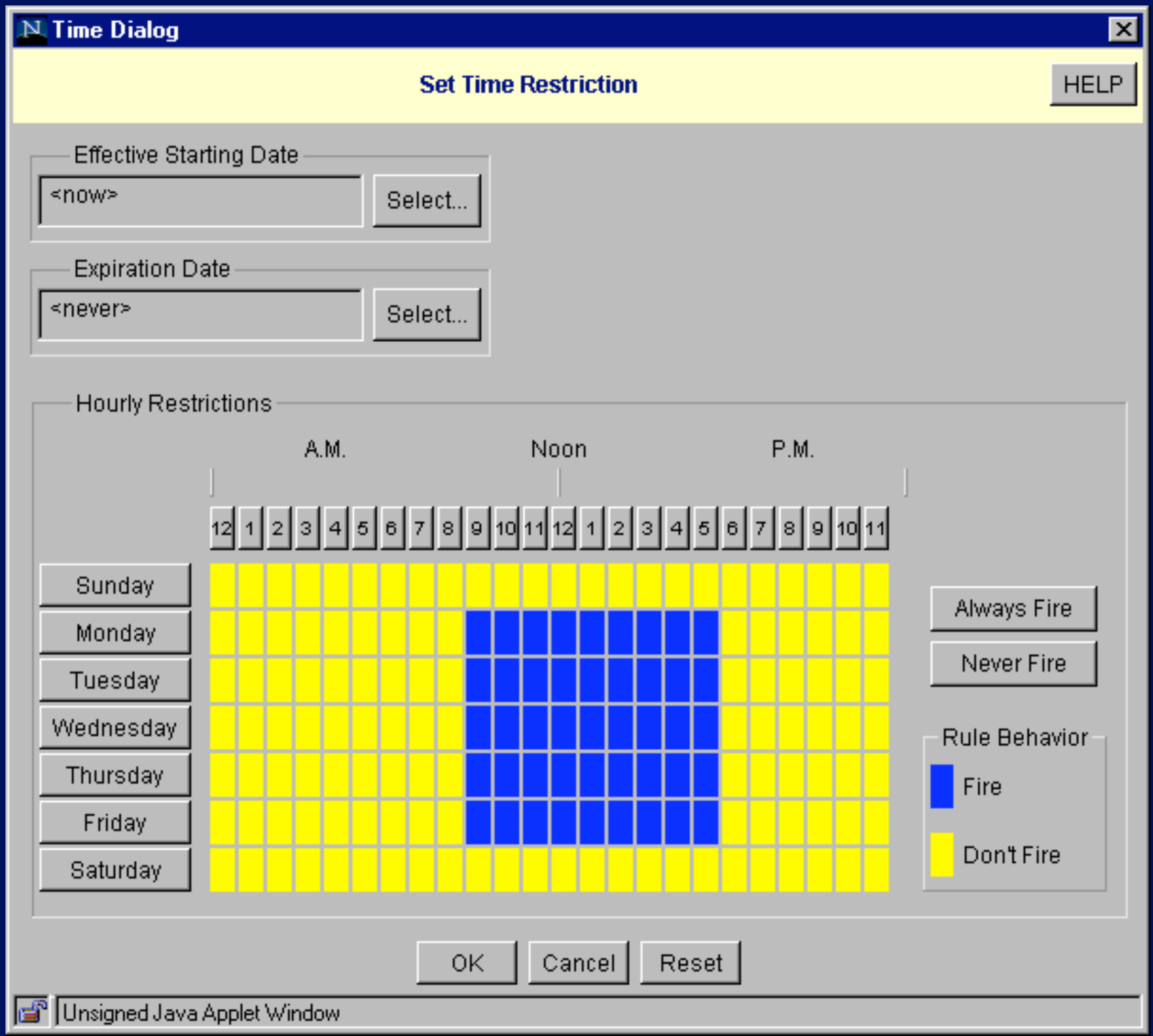




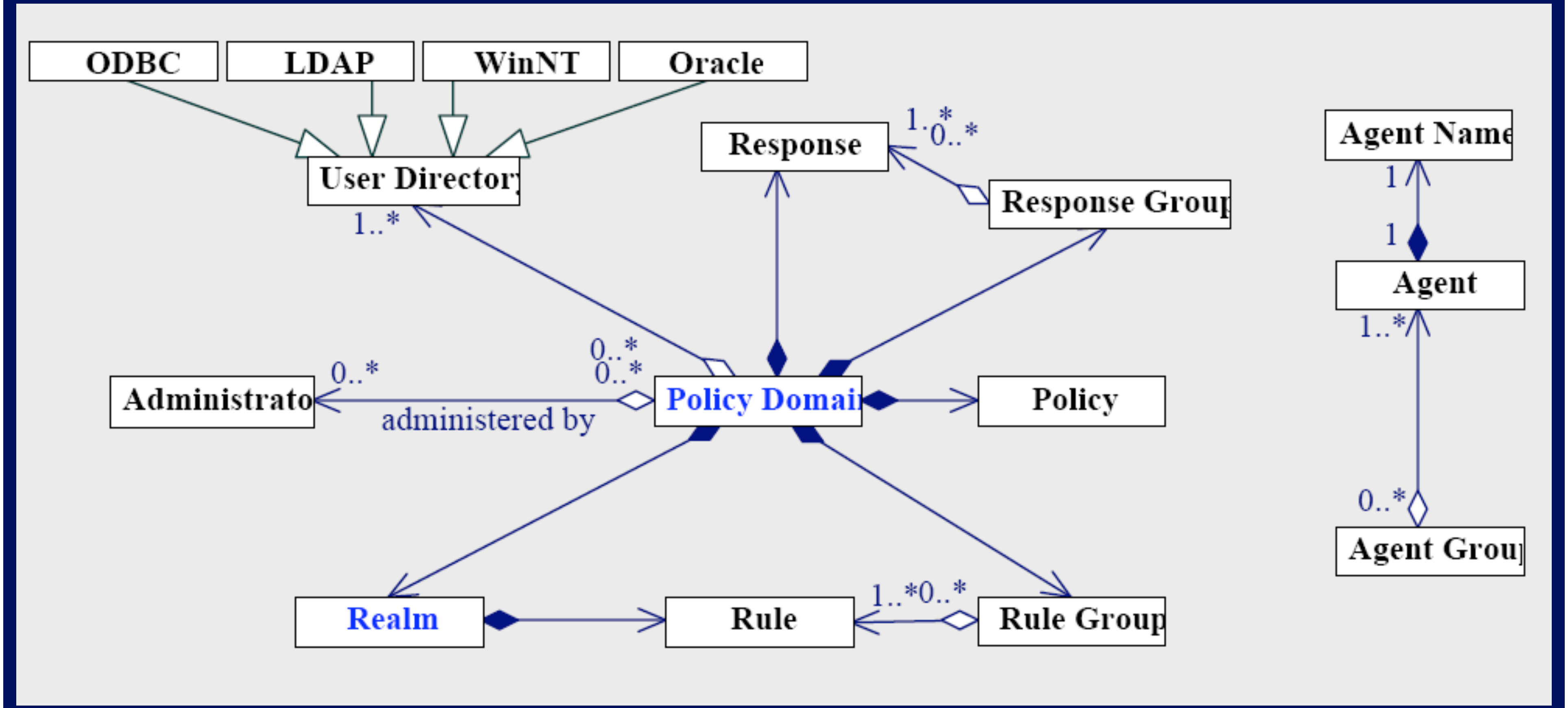
then continues with simple forms to fill out …



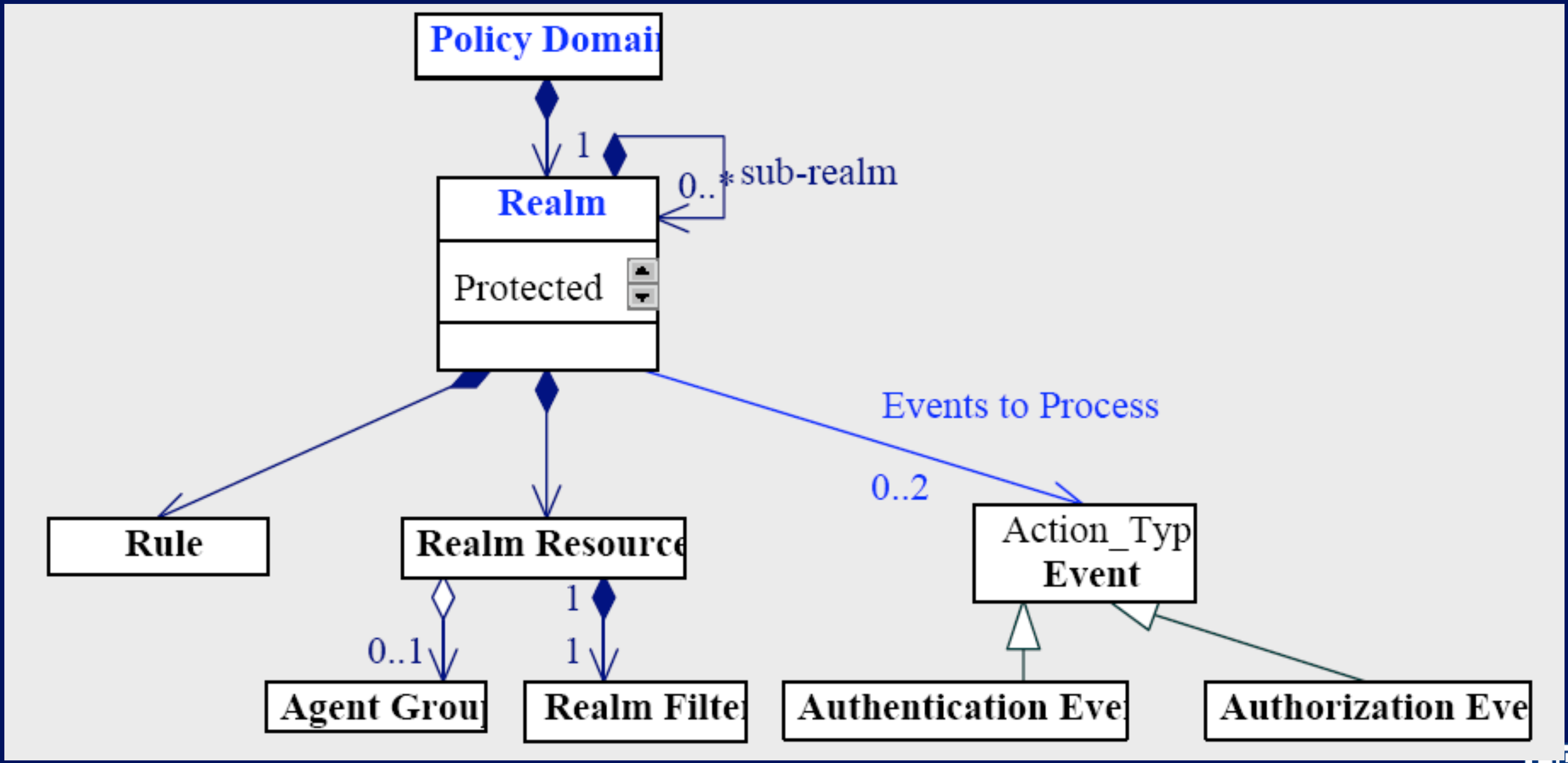
… or select


Time Dialog
Set Time Restriction
HELP
Effective Starting Date
<now>
Select...
Expiration Date
<never>
Select...
Hourly Restrictions
A.M.
Noon
P.M.
12 1 2 3 4 5 6 7 8 9 10 11 12 1 2 3 4 5 6 7 8 9 10 11
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Always Fire
Never Fire
Rule Behavior
Fire
Don't Fire
OK
Cancel
Reset
Unsigned Java Applet Window

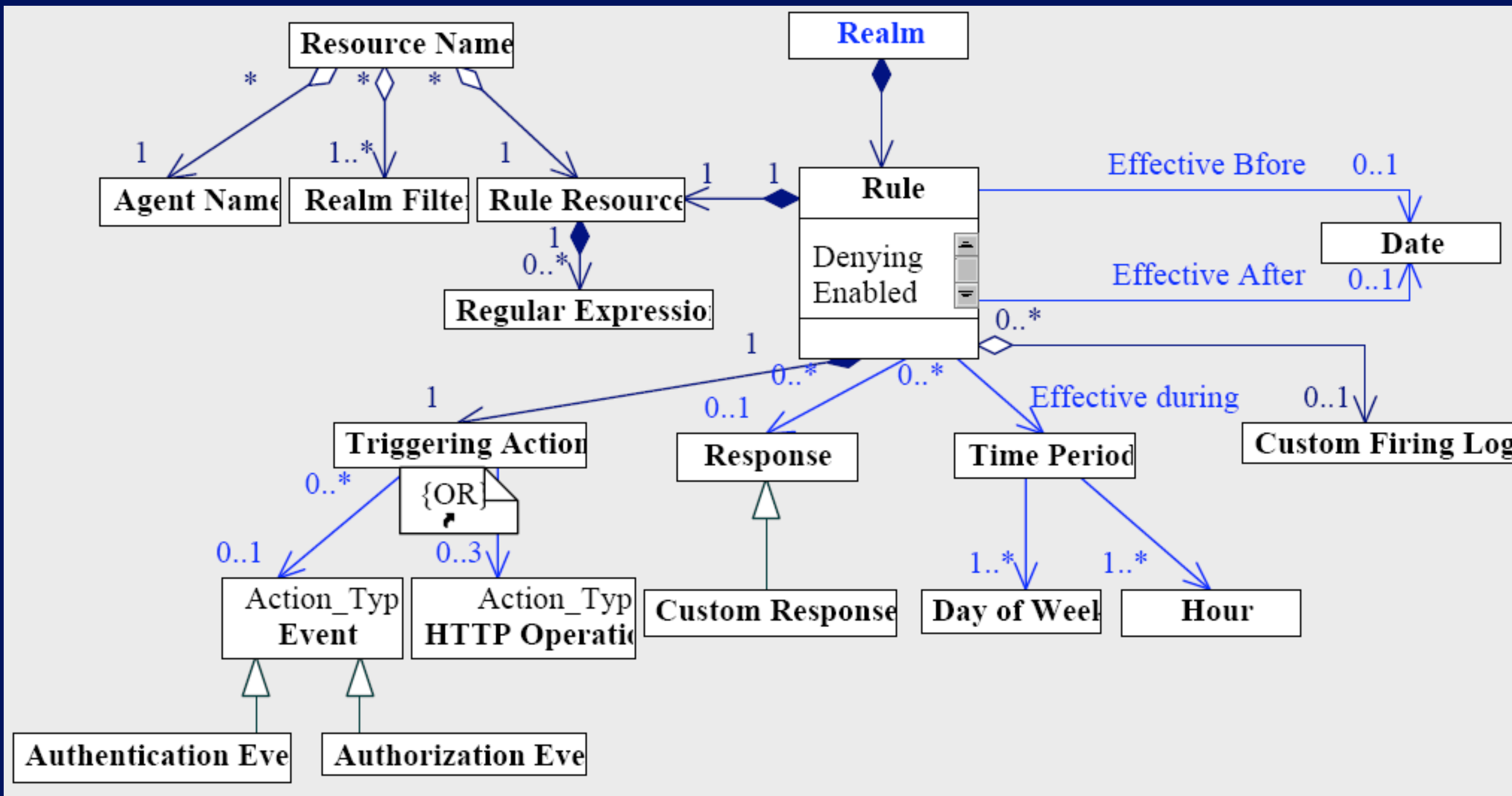
but the mental model is complex



… and even more …

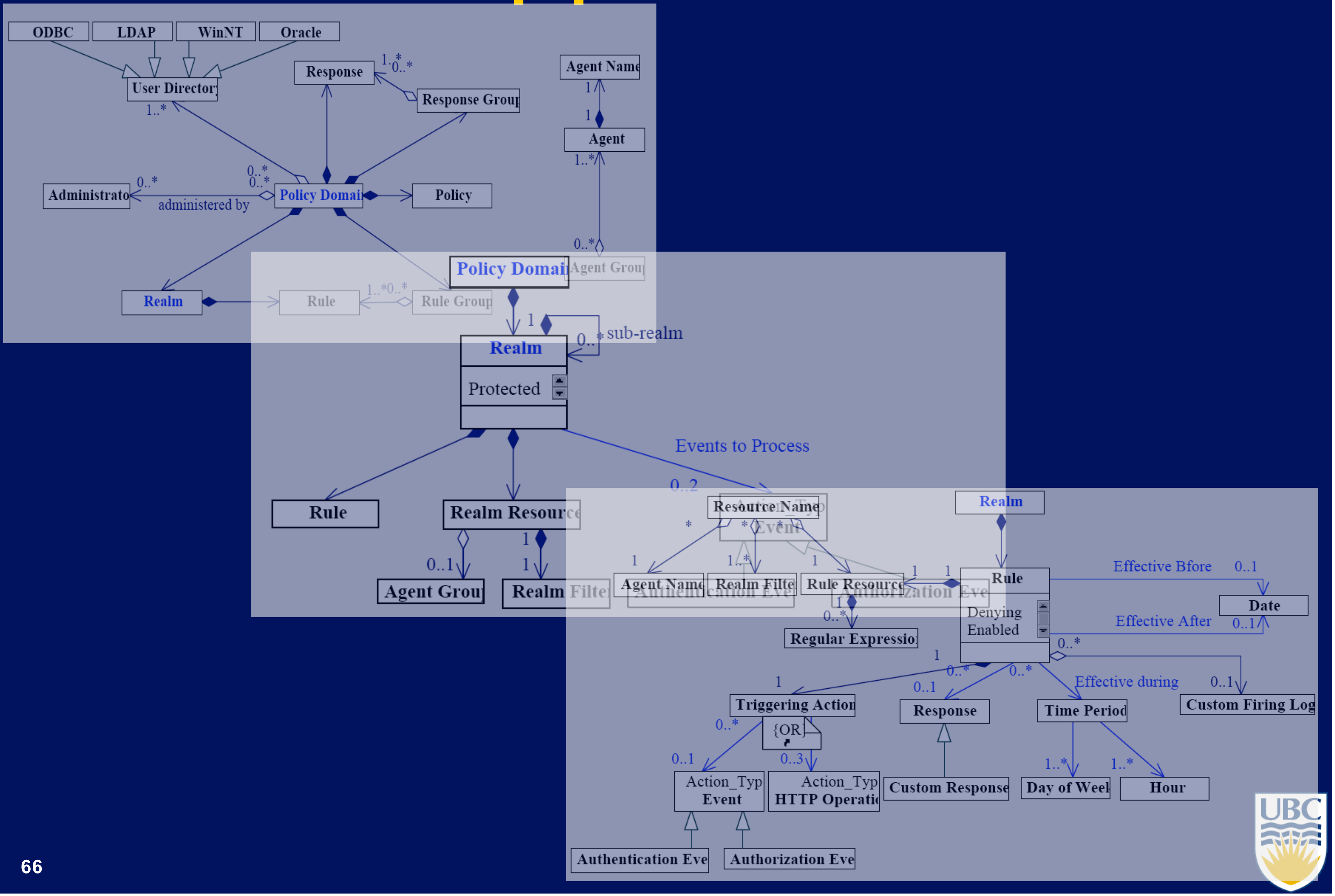


… complex



Resource Name
*
*
*
1
1..*
1
Agent Name
Realm Filte
Rule Resource
1
0..*
Regular Expressio
Realm
Rule
Denying
Enabled
1
1
Effective Bfore
0..1
Date
Effective After
0..1
0..*
1
0..*
0..*
1
0..1
Effective during
0..1
Triggering Action
Response
Time Period
Custom Firing Log
{OR}
0..*
0..1
0..3
Action_Typ
Event
Action_Typ
HTTP Operatic
Custom Response
1..*
1..*
Day of Weel
Hour
Authentication Eve
Authorization Eve

hard to map policies to models

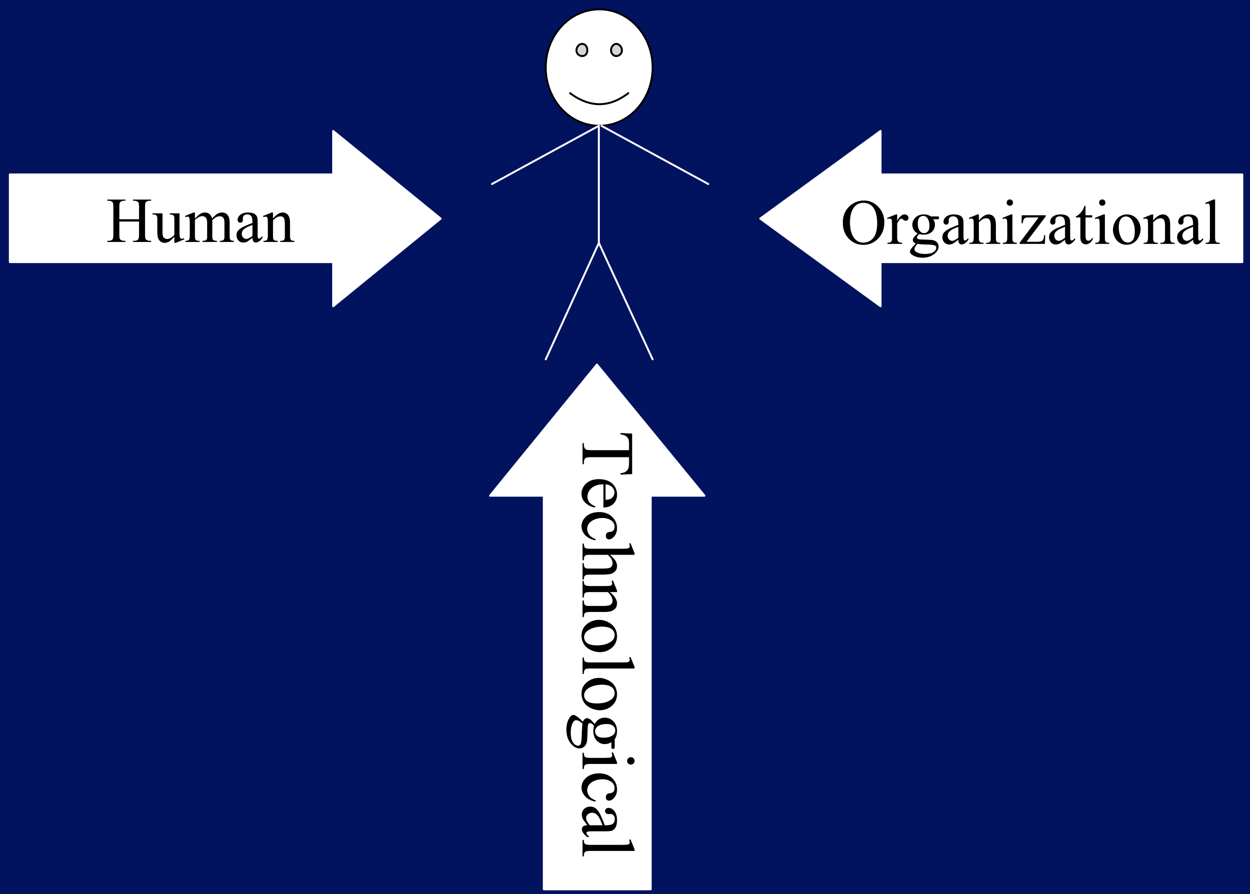


so what?

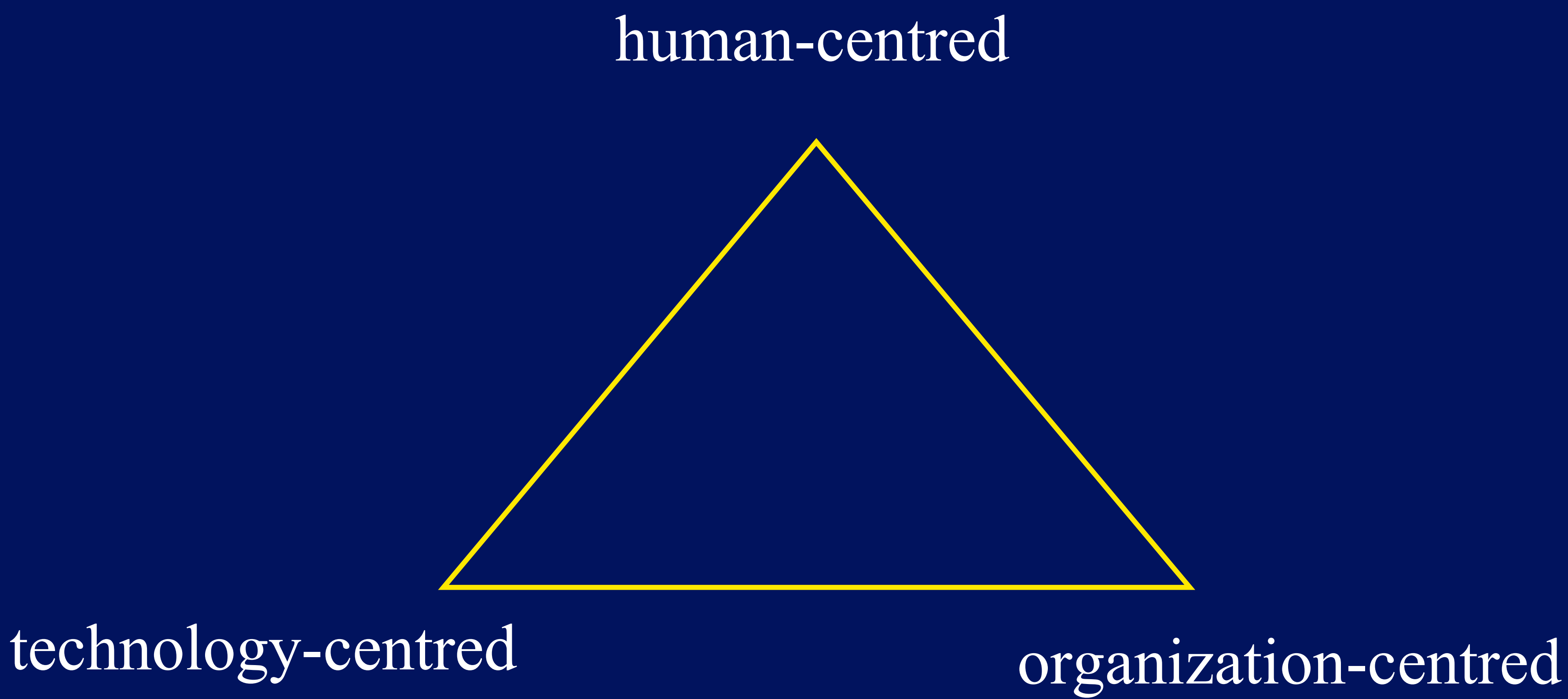
- steep learning curve
- hard to fit real world into the model
- easy to make costly mistakes
  - "friendly" DoS
  - inadvertent hard to catch config. vulnerabilities
- hard to test
  - expensive to test required scenarios
  - no "what if" scenarios to test before changing
  - hard to perform complete testing
- motivates users and admins to circumvent security

approach

administrators in the epicentres



approach



human-centred

better means for

1. visualizing the state of the security mechanisms
2. providing feedback to security admins
   - “what if” scenarios
   - safe staging playgrounds
   - tests of properties of the security state
3. support for cognitive models of system security

organization-centred

- patterns of communication between different parts of the organization and admins
- offload certain tasks from the admins

technology-centred

accommodate security technology to human and organizational needs

possible examples

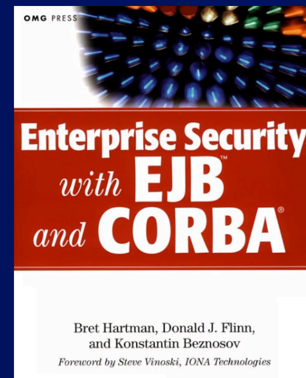
- self-administration
- domain-specific access control models and languages
- flexible and reconfigurable policy engines

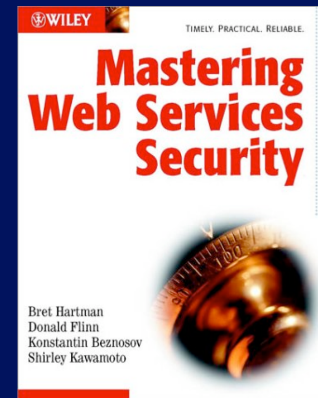
work plan

1. pilot studies to fine-tune study plans
2. inventories and an initial analysis through field research with industry
3. development of models
   - human, organizational, technological
4. design of techniques and methodologies
5. validation and evaluation of the project's key results
   - sample admin tools

team

- Dr. Konstantin Beznosov
  - Assist. Prof., ECE, UBC
  - 5 years of industry



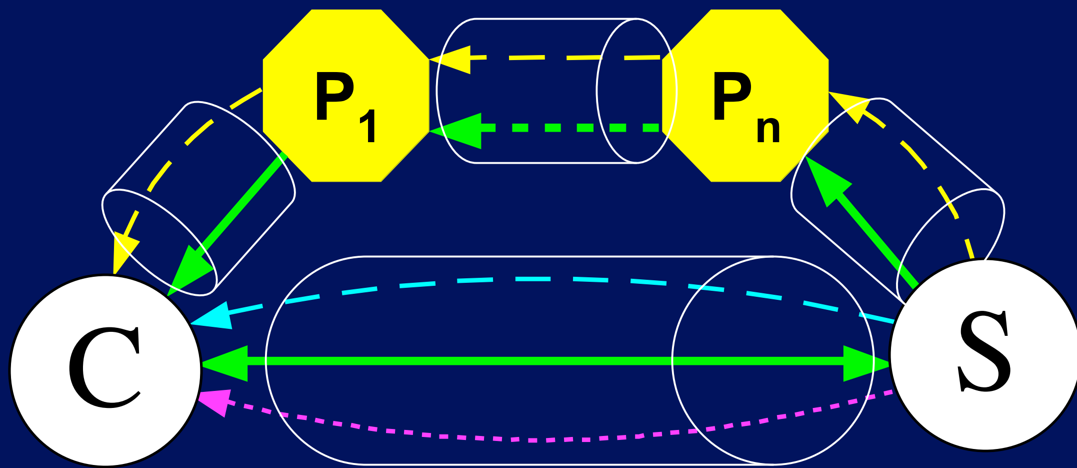


- Dr. Sidney Fels
  - Assoc. Prof., ECE, UBC
  - New interfaces design

- Dr. Brian Fisher
  - Assoc. Prof. of Interactive Arts and Technology, SFU
  - Adjunct Professor in MIS and CS, UBC
  - cognitive science-based interaction design

- Dr. Lee Iverson
  - Assist. Prof., ECE, UBC
  - information visualization and information systems
  - collaboration infrastructures

Direction: Network Security
Project: multiple-channel SSL



- end-to-end security with partially trusted proxies
- selective data protection