Fabric

A Platform for Secure Distributed Computation and Storage

Jed Liu Michael D. George K. Vikram
Xin Qi Lucas Waye Andrew C. Myers

Department of Computer Science
Cornell University

22nd ACM SIGOPS Symposium on Operating Systems Principles
14 October 2009

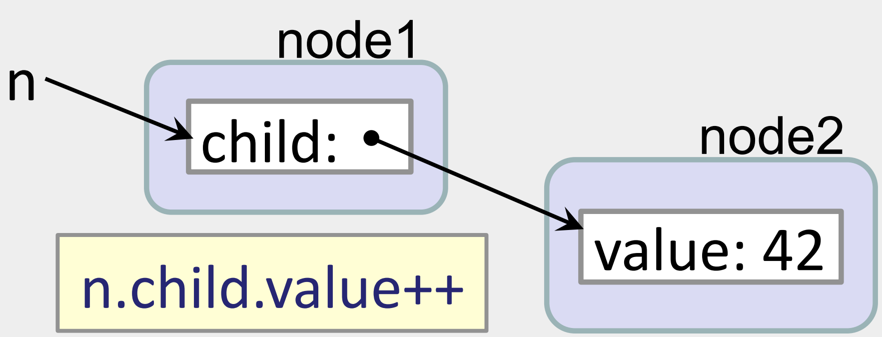
The Web is Not Enough

- The Web: decentralized information-sharing
- Limitations for integrating information
  - Medicine, finance, government, military, …

Is there a principled way to build federated applications while guaranteeing security and consistency?

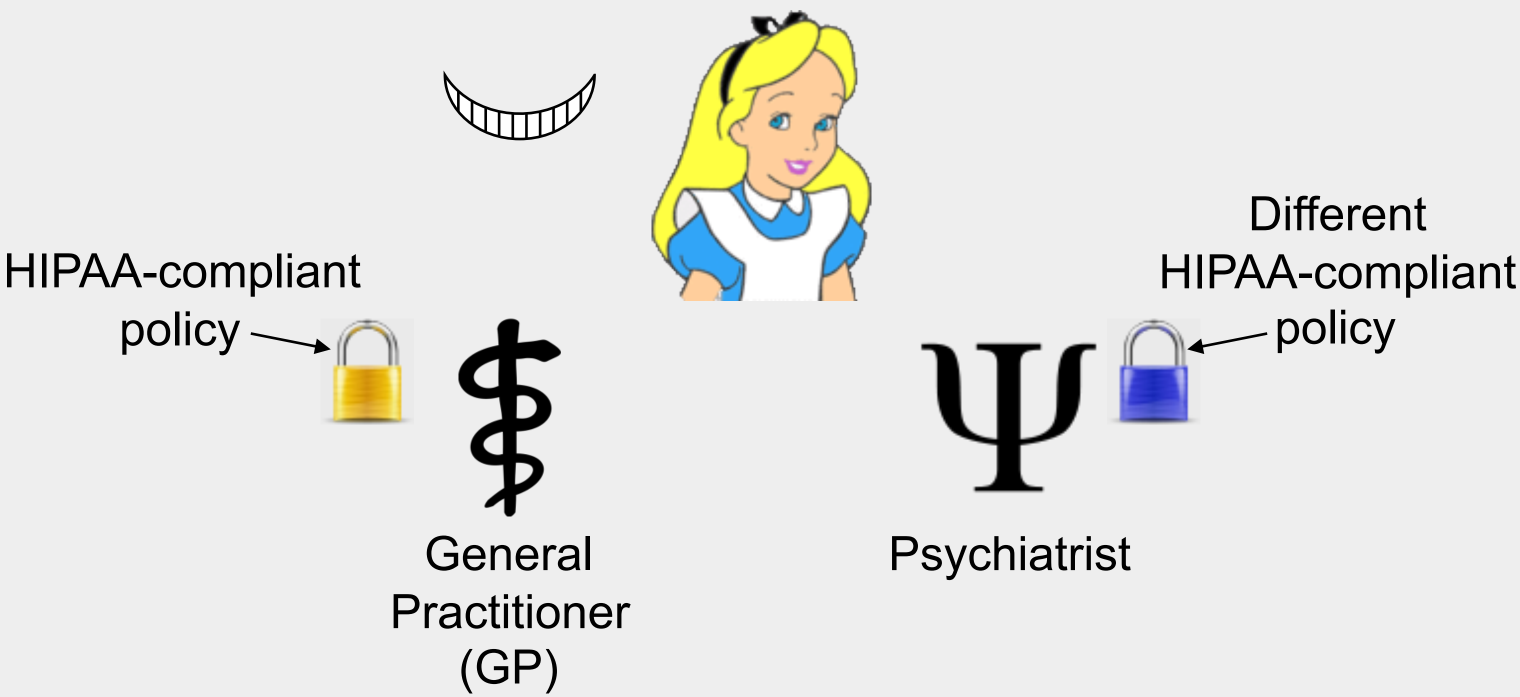
Fabric: A System and a Language

- Decentralized system for securely sharing information and computation
- All information looks like an ordinary program object
- Objects refer to each other with references
  - Any object can be referenced uniformly from anywhere
  - References can cross nodes and trust domains
  - All references look like ordinary object pointers

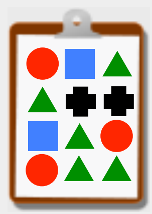


Compiler and runtime enforce security and consistency despite distrust

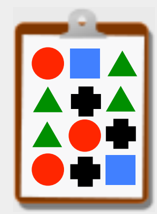
Fabric Enables Federated Sharing



Fabric Enables Federated Sharing

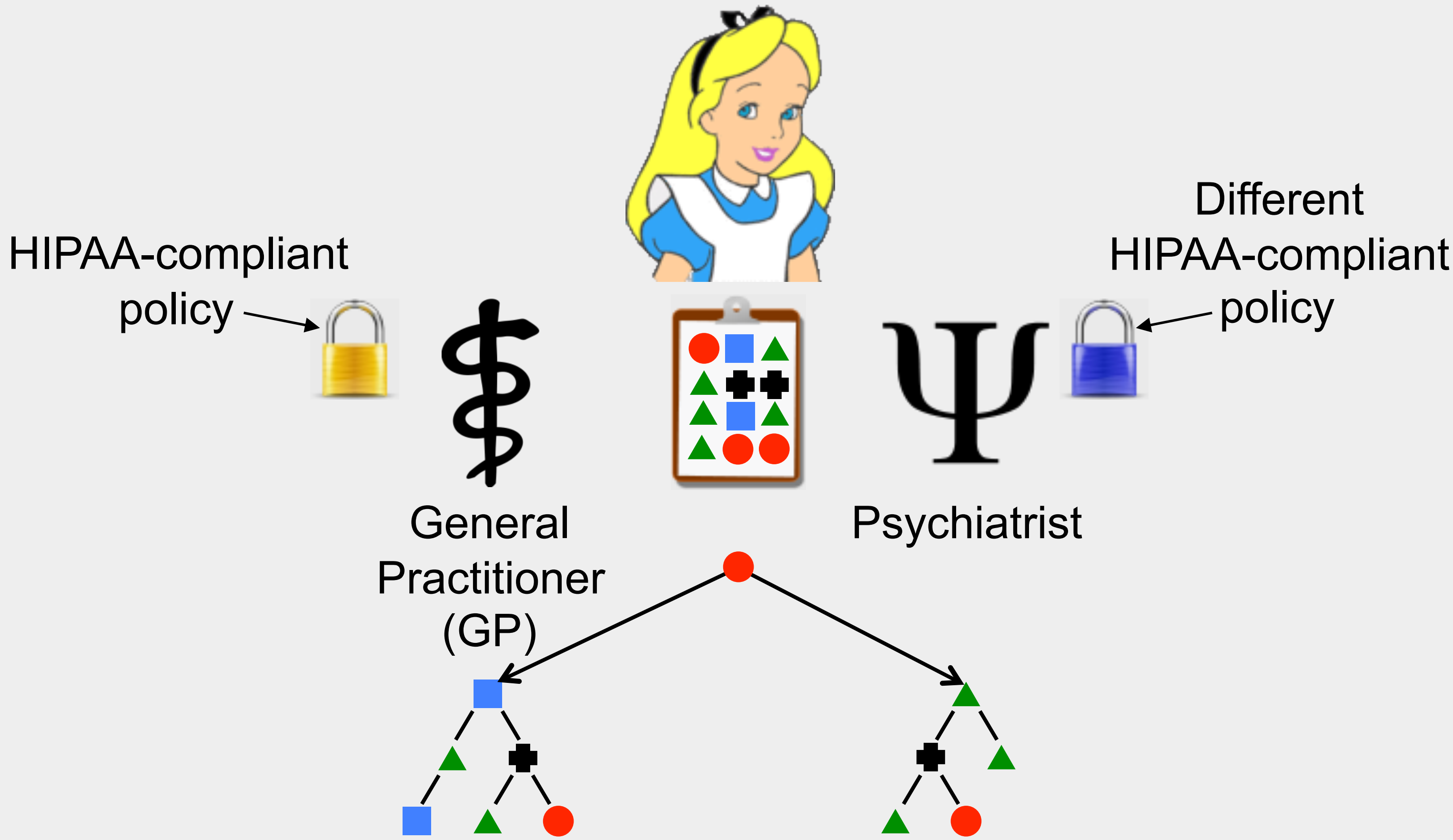


General Practitioner (GP)

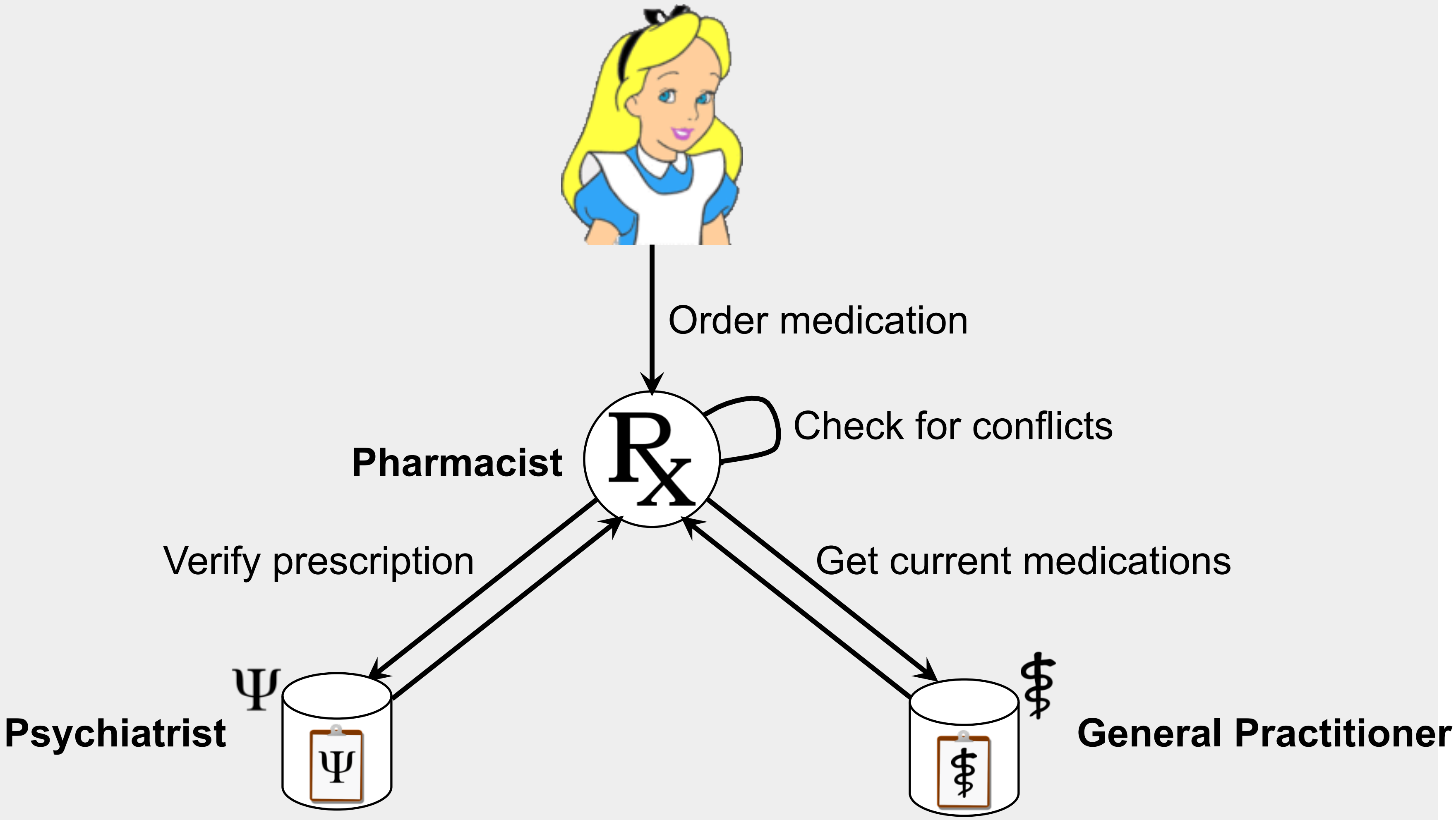


Psychiatrist

Fabric Enables Federated Sharing



Example: Filling a Prescription



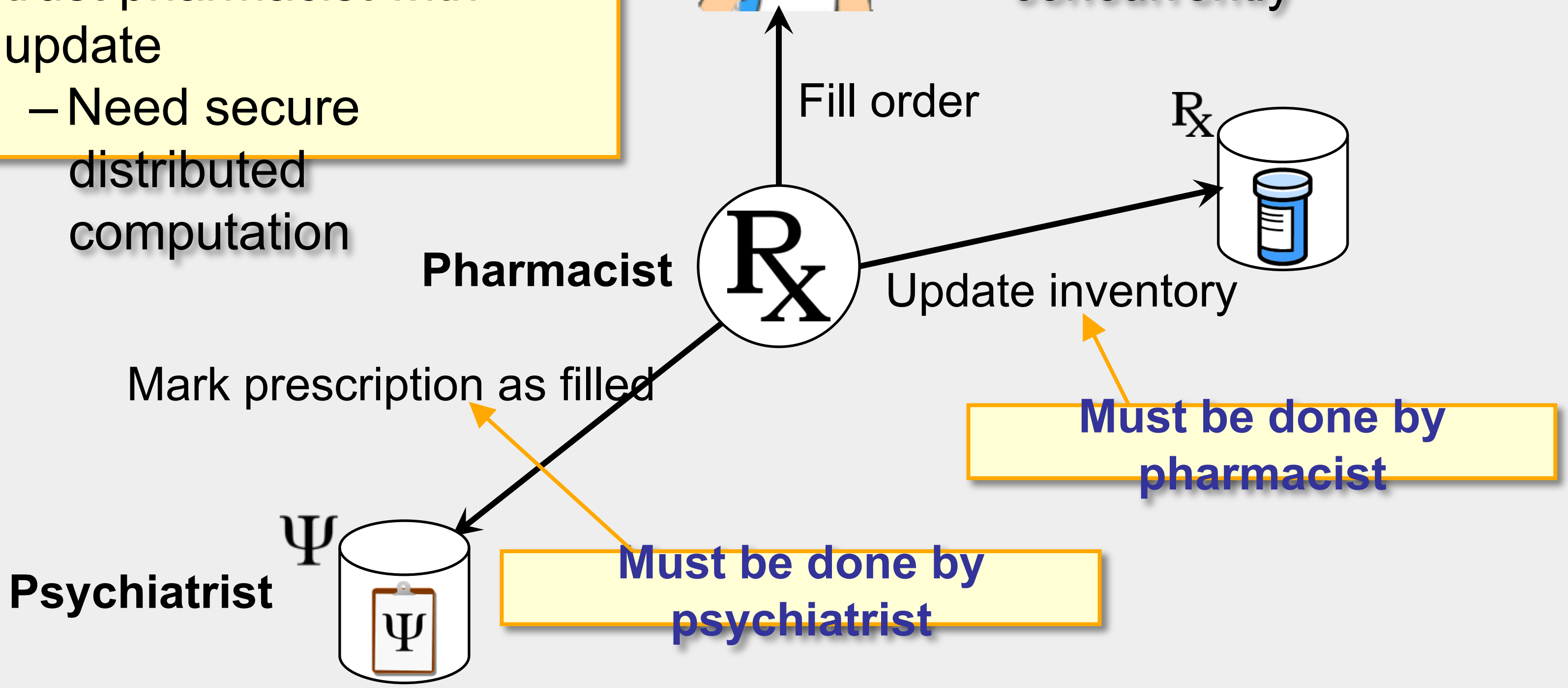
Example: Filling a Prescription

Security issues

- Pharmacist shouldn't see entire record
- Psychiatrist doesn't fully trust pharmacist with update
  - Need secure distributed computation

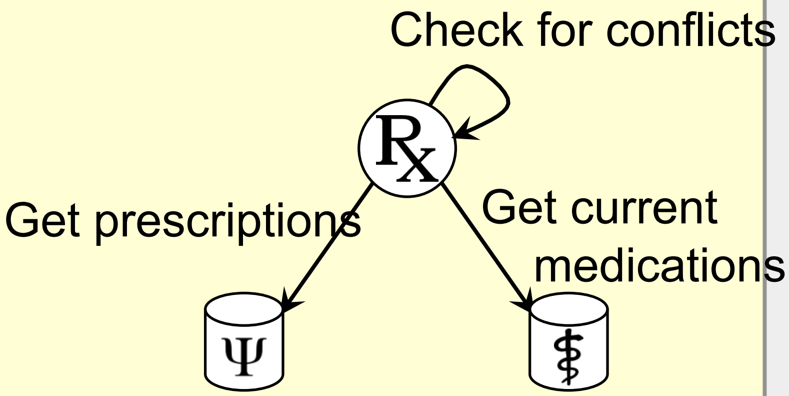
Consistency issues

- Need atomicity
- Doctors might be accessing medical record concurrently



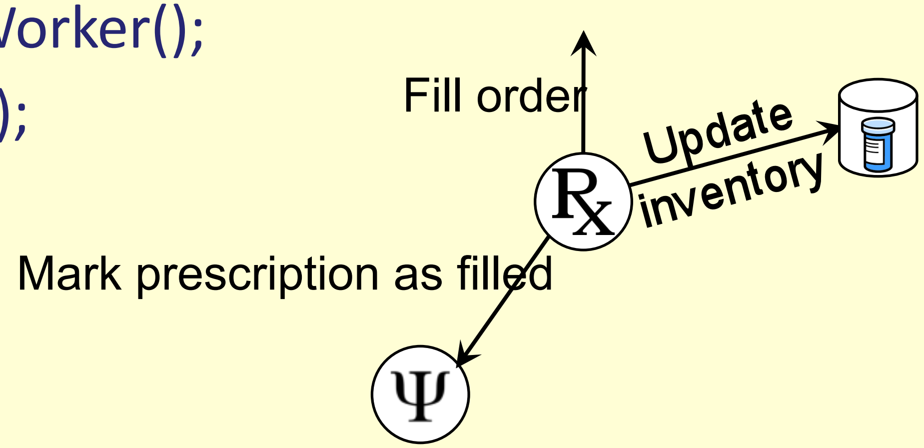
Pharmacy Example in Fabric

```
Order orderMed(PatRec psyRec, PatRec gpRec, Prescription p) {

    if (!psyRec.hasPrescription(p)) return Order.INVALID;
    if (isDangerous(p, gpRec.getMeds())) return Order.DANGER;

}
```



Pharmacy Example in Fabric

```
Order orderMed(PatRec psyRec, PatRec gpRec, Prescription p) {
  atomic {
    if (!psyRec.hasPrescription(p)) return Order.INVALID;
    if (isDangerous(p, gpRec.getMeds())) return Order.DANGER;

    Worker psy = psyRec.getWorker();
    psyRec.markFilled@psy(p);
    updateInventory(p);
    return Order.fill(p);
  }
}
```



A High-Level Language

```
Order orderMed(PatRec psyRec, PatRec gpRec, Prescription p) {
  atomic {
    if (!psyRec.hasPrescription(p)) return Order.INVALID;
    if (isDangerous(p, gpRec.getMeds())) return Order.DANGER;

    Worker psy = psyRec.getWorker();
    psyRec.markFilled@psy(p);
    updateInventory(p);
    return Order.fill(p);
  }
}
```

Java with:

- Remote calls
- Nested transactions (atomic blocks)
- Label annotations for security (elided)

A High-Level Language

```
Order orderMed(PatRec psyRec, PatRec gpRec, Prescription p) {
  atomic {
    if (!psyRec.hasPrescription(p)) return Order.INVALID;
    if (isDangerous(p, gpRec.getMeds())) return Order.DANGER;

    Worker psy = psyRec.getWorker();
    psyRec.markFilled@psy(p);
    updateInventory(p);
    return Order.fill(p);
  }
}
```

- All objects accessed uniformly regardless of location
- Objects fetched as needed
- Remote calls are explicit

Run-time system requirement:

- **Secure transparent data shipping**

Remote Calls

```
Order orderMed(PatRec psyRec, PatRec gpRec, Prescription p) {
  atomic {
    if (!psyRec.hasPrescription(p)) return Order.INVALID;
    if (isDangerous(p, gpRec.getMeds())) return Order.DANGER;

    Worker psy = psyRec.getWorker();
    psyRec.markFilled@psy(p);
    updateInventory(p);
    return Order.fill(p);
  }
}
```

Remote call — pharmacist runs method at psychiatrist's node

Run-time system requirements:

- Secure transparent data shipping
- **Secure remote calls**

Federated Transactions

```
Order orderMed(PatRec psyRec, PatRec gpRec, Prescription p) {
  atomic {
    if (!psyRec.hasPrescription(p)) return Order.INVALID;
    if (isDangerous(p, gpRec.getMeds())) return Order.DANGER;

    Worker psy = psyRec.getWorker();
    psyRec.markFilled@psy(p);
    updateInventory(p);
    return Order.fill(p);
  }
}
```

Federated transaction — spans multiple nodes & trust domains

Remote call — pharmacist runs method at psychiatrist's node

Run-time system requirements:

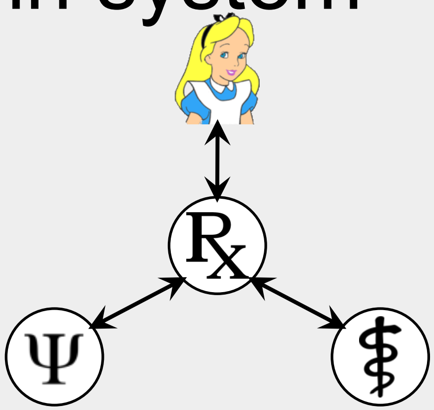
- Secure transparent data shipping
- Secure remote calls
- **Secure federated transactions**

Fabric Security Model

- Decentralized system – anyone can join
- What security guarantees can we provide?
- **Decentralized security principle:**

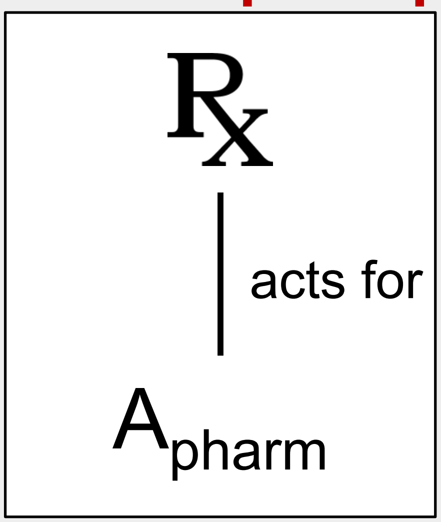
  You can't be hurt by what you don't trust

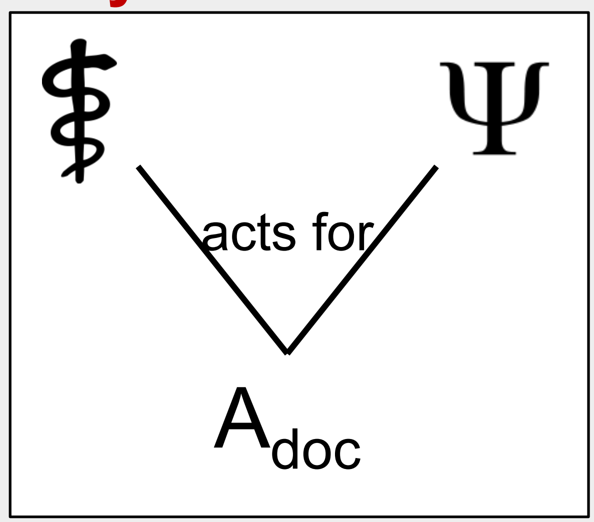
- Need notion of "you" and "trust" in system and language
  - **Principals** and **acts-for**



Principals and Trust in Fabric

- **Principals** represent users, nodes, groups, roles
- Trust delegated via **acts-for**
  - "Alice acts-for Bob" means "Bob trusts Alice"
  - Like "speaks-for" [LABW91]
  - Generates a **principal hierarchy**





Trust Management

- Fabric principals are objects

```
class Principal {
  boolean delegatesTo(principal p);
  void addDelegatesTo(principal p) where caller (this);
  …
}
```

Determines whether *p* acts for this principal

Caller must have authority of this principal

- Explicit trust delegation via method calls

```
// Adds "Alice acts-for Bob" to principal hierarchy
bob.addDelegatesTo(alice)
```

  – Compiler and run-time ensure that caller has proper authority

Security Labels in Fabric

- Based on Jif programming language [M99]
- Decentralized label model [ML98]
  - Labels specify security policies to be enforced

Confidentiality: Alice $\rightarrow$ Bob — Alice permits Bob to read

Integrity: Alice $\leftarrow$ Bob — Alice permits Bob to write

```
class Prescription {
  Drug{Psy→A_pharm; Psy←Psy} drug;
  Dosage{Psy→A_pharm; Psy←Psy} dosage;
  … }
```

- Compiler and run-time system ensure that policies are satisfied

Security Labels in Fabric

- Based on Jif programming language [M99]
- Decentralized label model [ML98]
  - Labels specify security policies to be enforced

Confidentiality: Alice → Bob   Alice permits Bob to read

Integrity: Alice ← Bob   Alice permits Bob to write

```
class Prescription {
  Drug{Psy → A_pharm; Psy ← Psy} d
  Dosage{Psy → A_pharm; Psy ← Psy}
  … }
```

Run-time system requirements:
- Secure transparent data shipping
- Secure remote calls
- Secure federated transactions
- **Enforcement of security labels**

- Compiler and run-time s policies are satisfied

Contributions

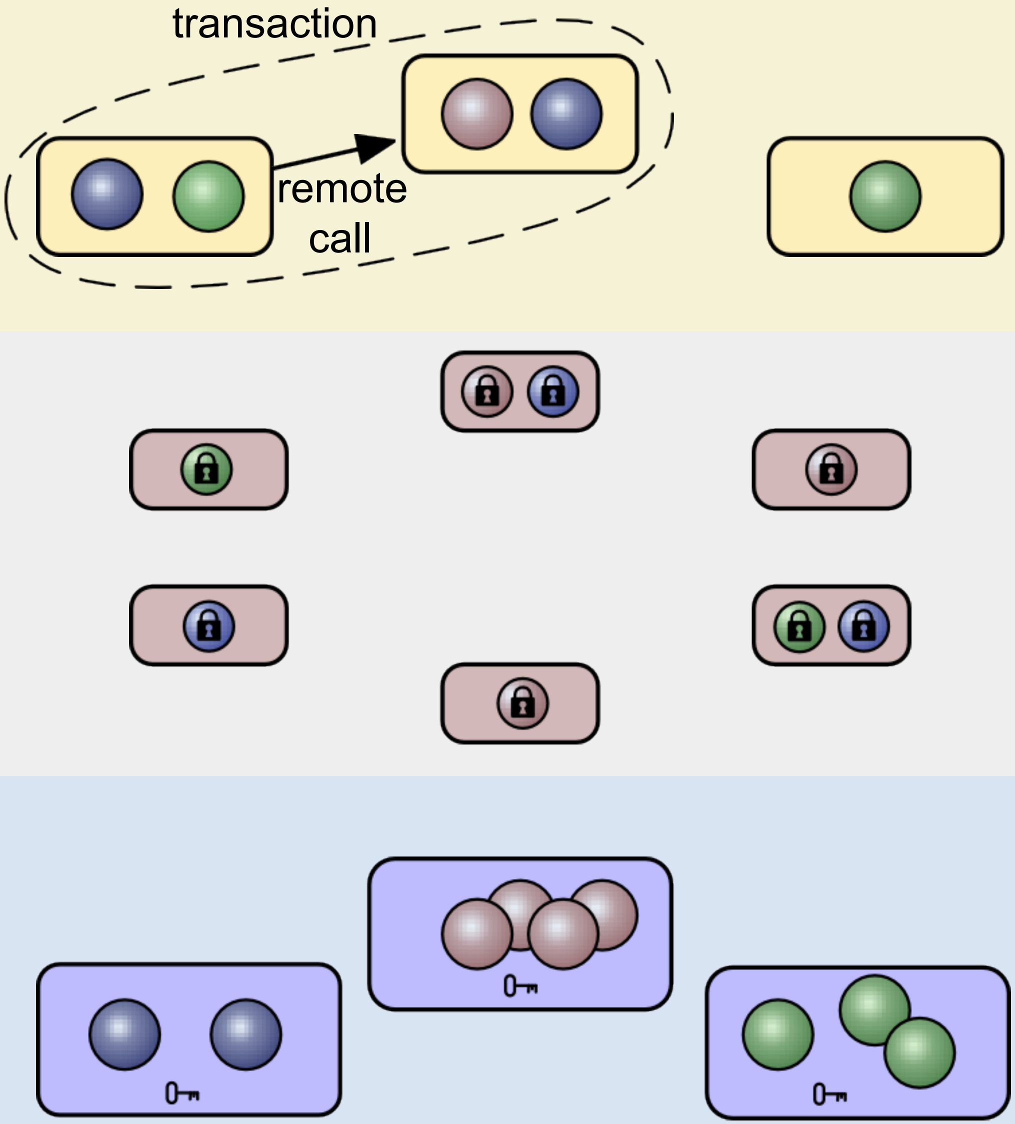
- Language combining:
  - Remote calls
  - Nested transactions
  - Security annotations
- System with:
  - Secure transparent data shipping
  - Secure remote calls
  - Secure federated transactions
  - Enforcement of security labels

Challenge: How to provide all these in the same system?

Fabric Run-Time System

- Decentralized platform for secure, consistent sharing of information and computation
  - Nodes join freely
  - No central control over security
- Nodes are principals
  - Root of trust
  - Authentication: X.509 certificates bind hostnames to principal objects

Fabric Architecture

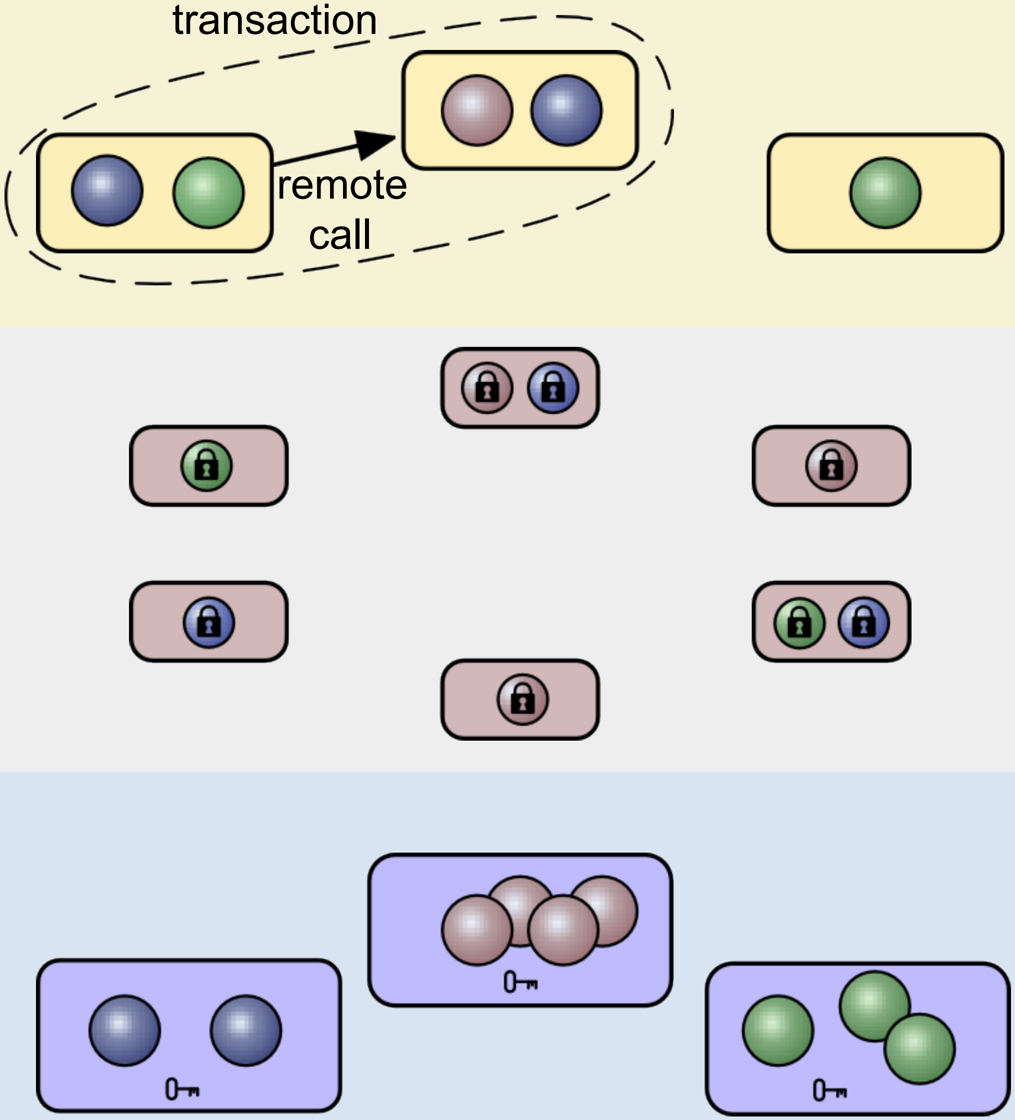


Worker nodes (Workers)

Dissemination nodes

Storage nodes (Stores)

Fabric Architecture

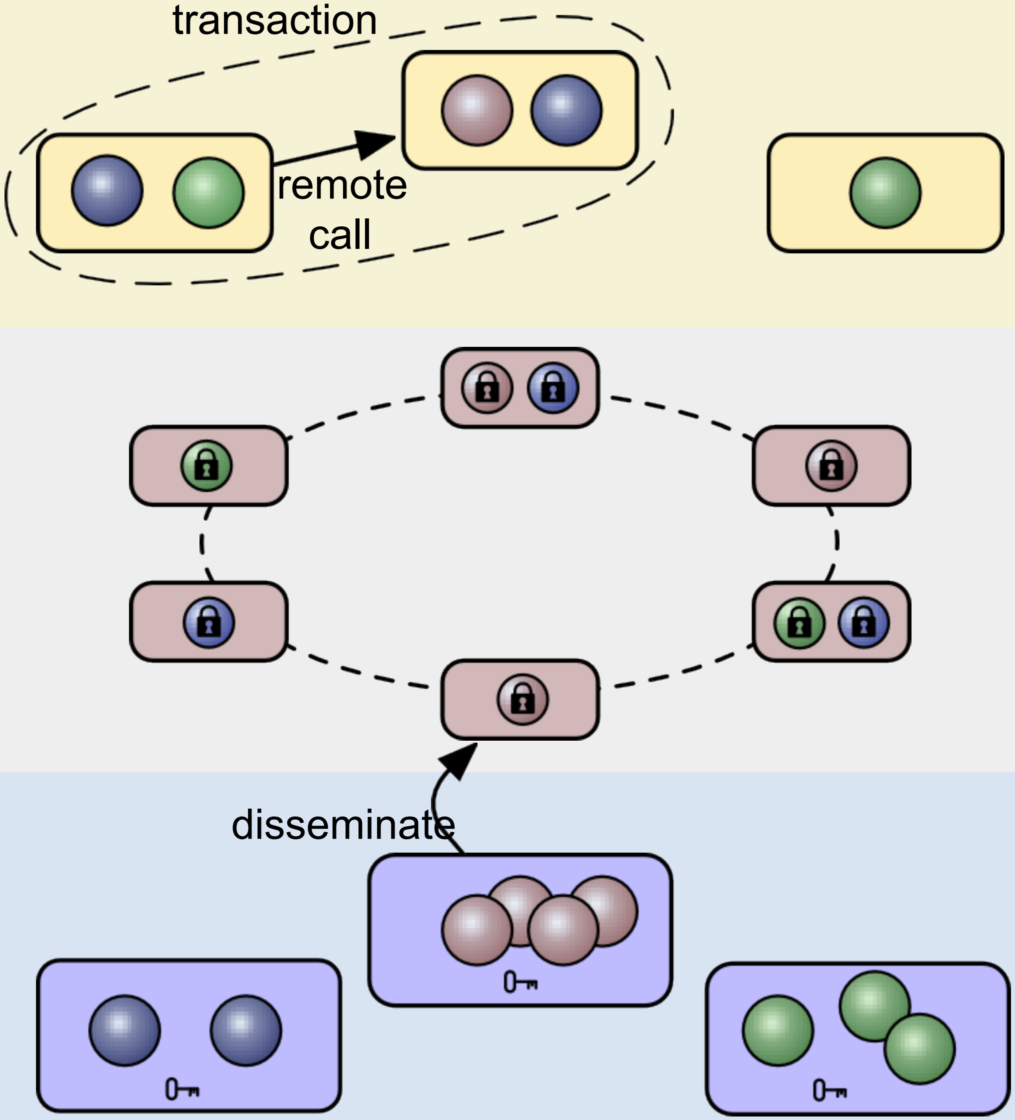


Worker nodes (Workers)

Dissemination nodes

- **Storage nodes** securely store persistent objects
- Each object specifies its own security policy, enforced by store

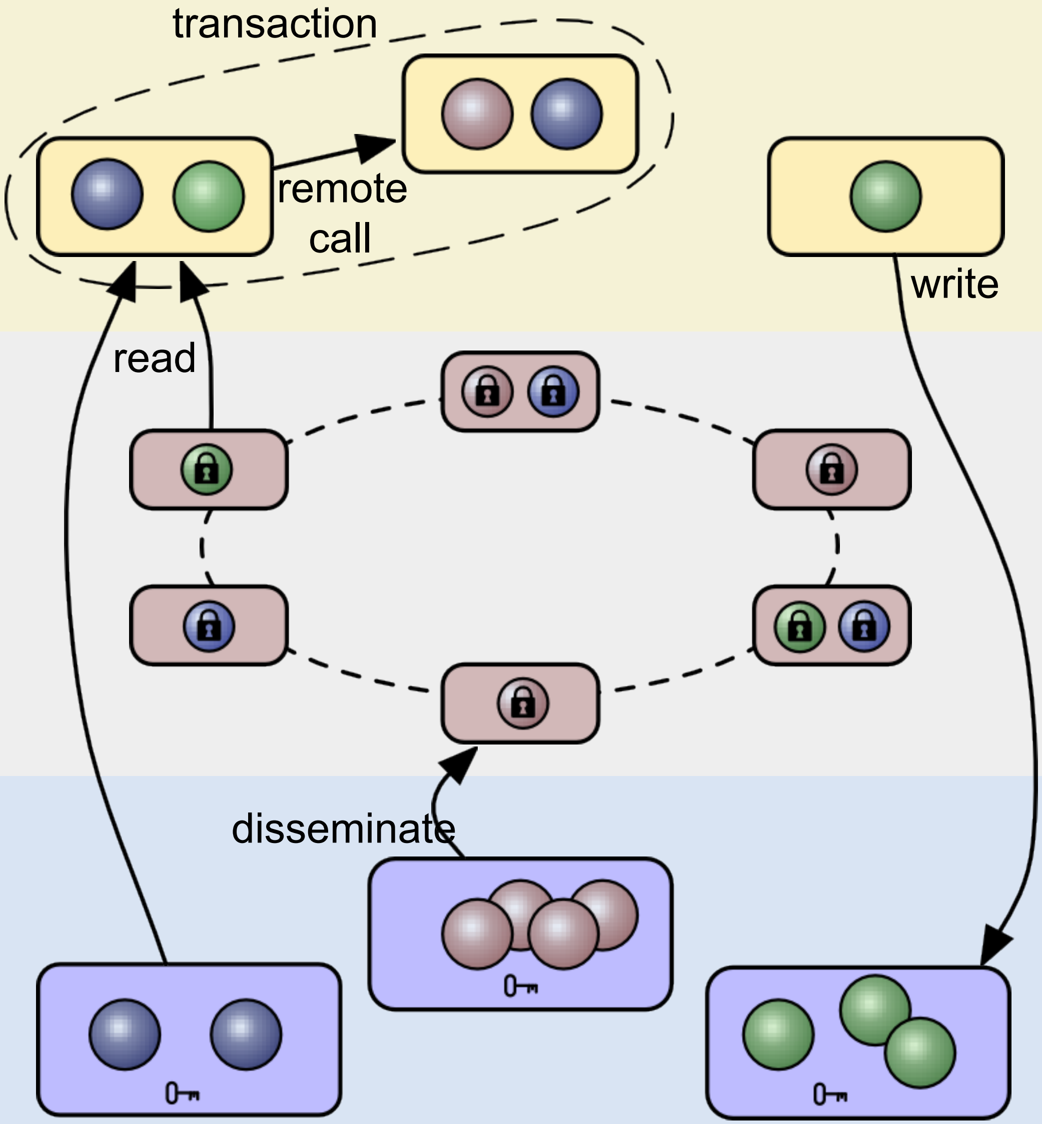
Fabric Architecture



Worker nodes (Workers)

- **Dissemination nodes** cache signed, encrypted objects in peer-to-peer distribution network for high availability
- **Storage nodes** securely store persistent objects
- Each object specifies its own security policy, enforced by store

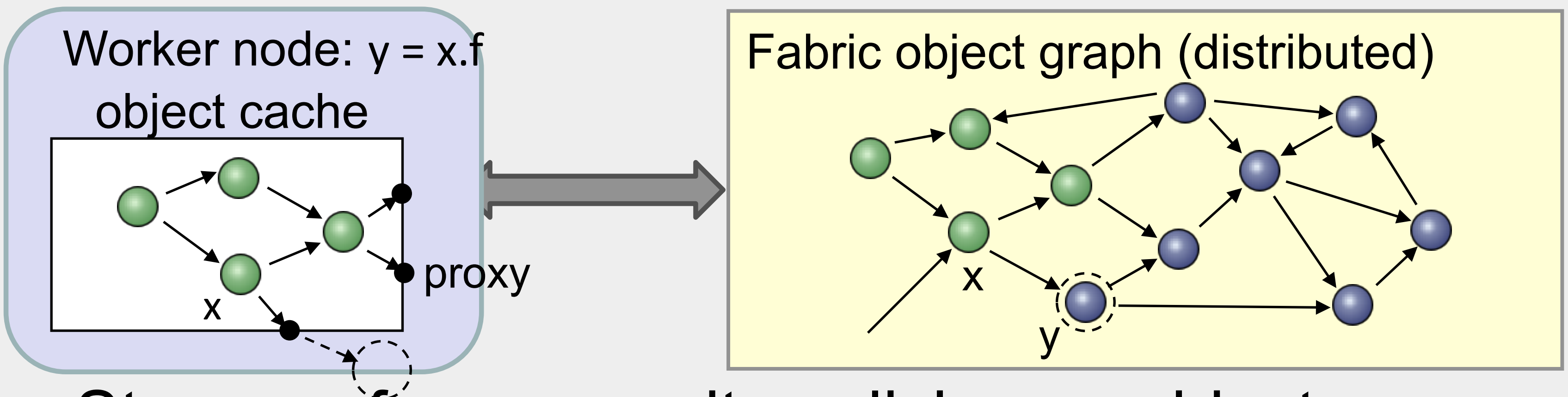
Fabric Architecture



- **Worker nodes** compute on cached objects
- Computation may be distributed across workers in **federated transactions**
- **Dissemination nodes** cache signed, encrypted objects in peer-to-peer distribution network for high availability
- **Storage nodes** securely store persistent objects
- Each object specifies its own security policy, enforced by store

Secure Transparent Data Shipping

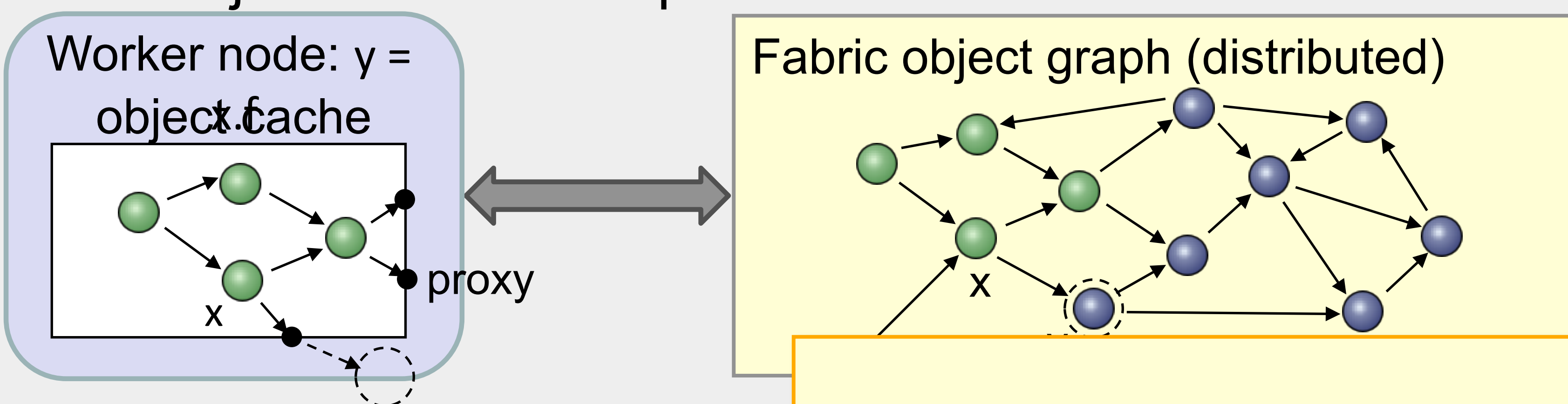
- Illusion of access to arbitrarily large object graph
  - Workers cache objects
  - Objects fetched as pointers are followed out of cache



- Stores enforce security policies on objects
  - Worker can read (write) object only if it's trusted to enforce confidentiality (integrity)

Secure Transparent Data Shipping

- Illusion of access to arbitrarily large object graph
  - Workers cache objects
  - Objects fetched as pointers are followed out of cache



- Stores enforce security
  - Worker can read (write)
    enforce confidentiality (in

Run-time system requirements:

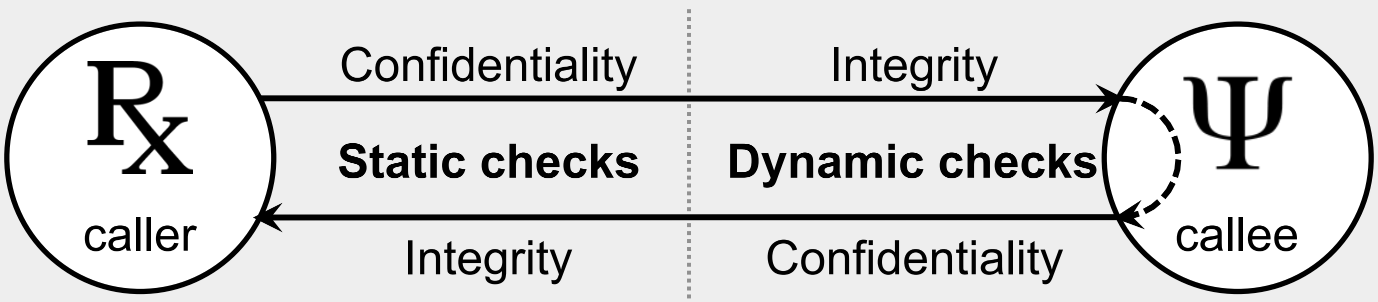
✓Secure transparent data shipping
- Secure remote calls
- Secure federated transactions

✓Enforcement of security labels

Secure Remote Calls

Is callee trusted to see call?
- Call itself might reveal private information
- Method arguments might be private

Is caller trusted to make call?
- Caller might not have sufficient authority to make call
- Method arguments might have been tampered with by caller

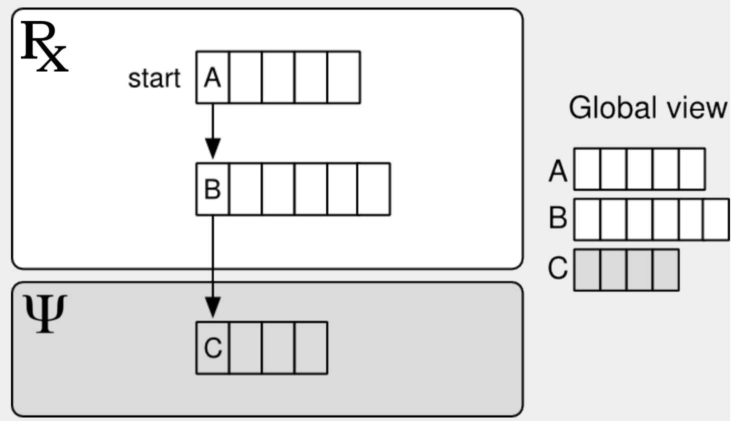


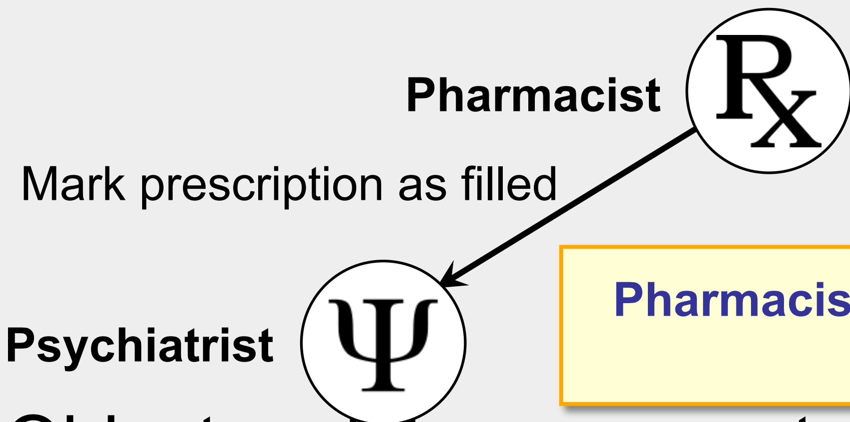
Is callee trusted to execute call?
- Call result might have been tampered with by callee

Is caller trusted to see result?
- Call result might reveal private information

Secure Federated Transactions

- Transactions can span multiple workers, cross trust domains
  - No single node trusted for entire log: distributed log structure





Pharmacist not trusted to log update

- Object updates propagated transparently and securely in multi-worker transactions

Also in the Paper...

- Dissemination of encrypted object groups
  - Key management to support this
- Writer maps for secure propagation of updates
- Hierarchical two-phase commit for federated transactions
- Interactions of transaction abort and information flow control
- Automatic ‘push’ of updated objects to dissemination layer
- In-memory caching of object groups at store
- Caching acts-for relationships at workers

Implementation

- Fabric prototype implemented in Java and Fabric
  - Total: 35 kLOC
  - Compiler translates Fabric into Java
    - 15 k-line extension to Jif compiler
    - Polyglot [NCM03] compiler extension
  - Dissemination layer: 1.5k-line extension to FreePastry
    - Popularity-based replication (à la Beehive [RS04])
  - Store uses BDB as backing store

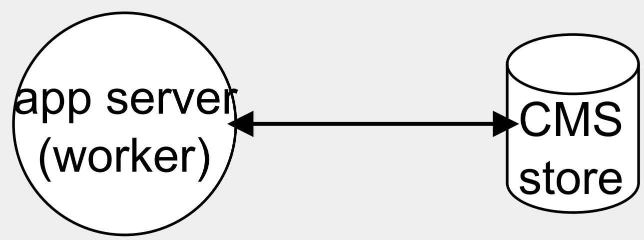
Overheads in Fabric

- Extra overhead on object accesses at worker
  - Run-time label checking
  - Logging reads and writes
  - Cache management (introduces indirection)
  - Transaction commit
- Overhead at store for reads and commits
- Ported non-trivial web app to evaluate performance

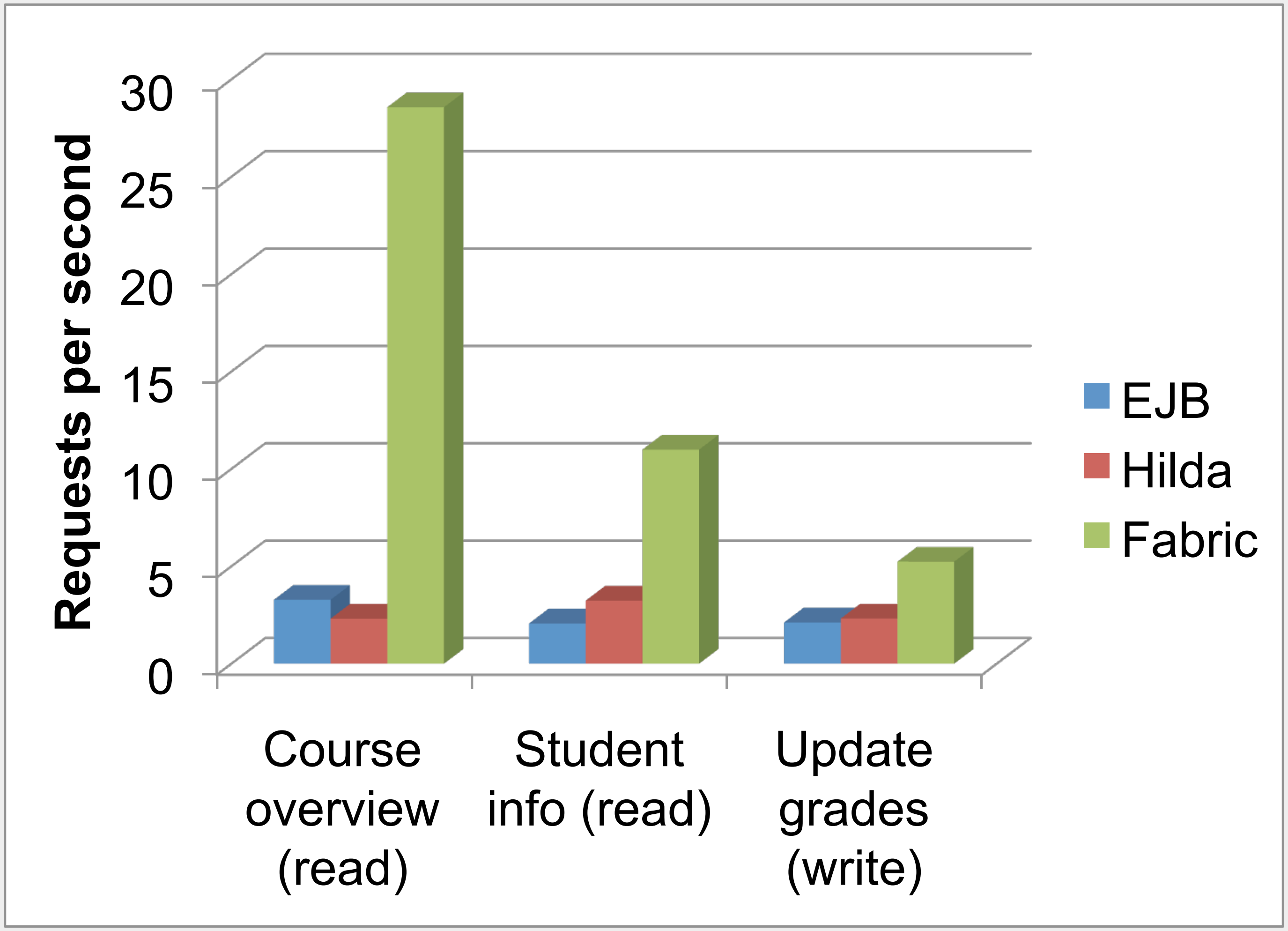
Cornell CMS Experiment

- Used at Cornell since 2004
  - Over 2000 students in over 40 courses
- Two prior implementations:
  - J2EE/EJB2.0
    - 54k-line web app with hand-written SQL
    - Oracle database

  - Hilda [YGG+07]
    - High-level language for data-driven web apps
- Fabric implementation



Performance Results



Summary

- Fabric is a platform for secure and consistent federated sharing
- Prototype implementation
- Contributions:
  - High-level language integrating information flow, transactions, distributed computation
  - Transparent data shipping and remote calls while enforcing secure information flow
  - New techniques for secure federated transactions: hierarchical commits, writer maps