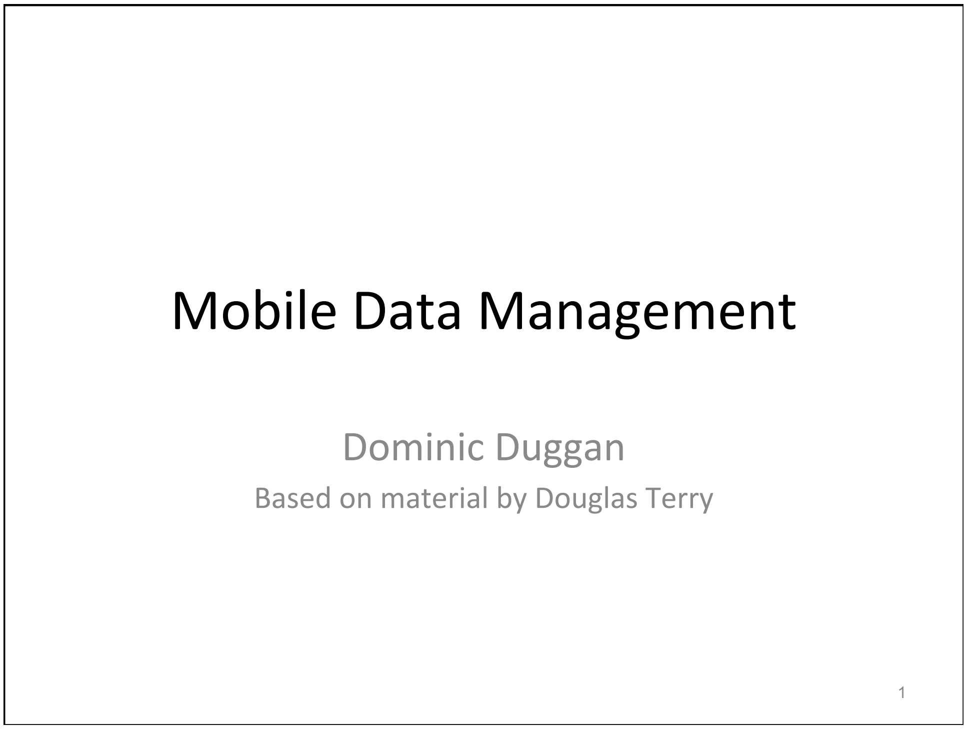
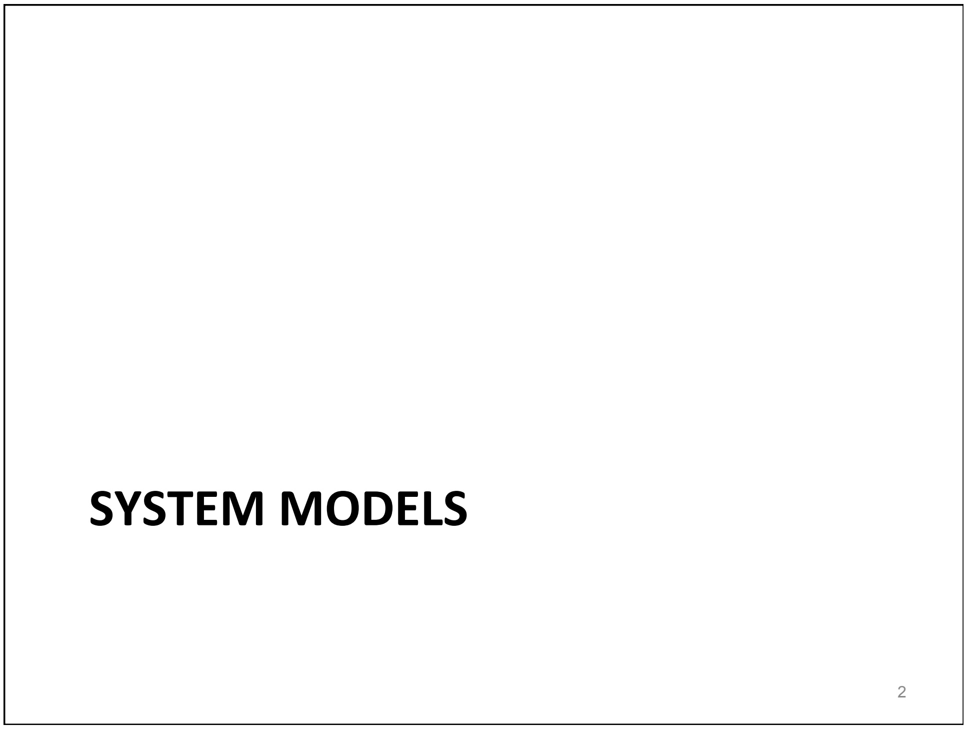


Mobile Data Management

Dominic Duggan

Based on material by Douglas Terry



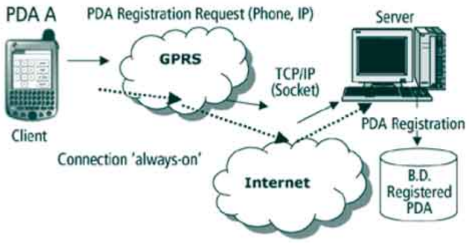
SYSTEM MODELS

System Models

- Issue: How to make data available to mobile devices
- Approaches:
  - Remote (server-based) data access
  - Device-master replication
  - Peer-to-peer replication
  - Publish-subscribe

Remote Data Access

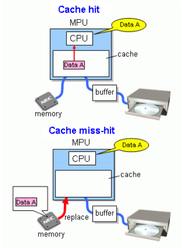
- Retrieve data from server (always!)
- Connectivity
- Battery life

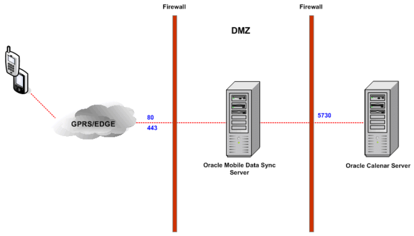


Device-Master Replication

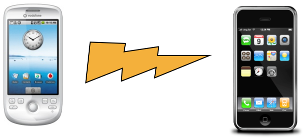
- Device-side caching
  - Ex: caching in CODA
  - On-demand caching
  - Hoarding/stashing

- Active, user-visible replica
  - Ex: syncing PDA, cell phone calendar
  - Weakened consistency requirements (stale data)
  - Potential for update conflicts





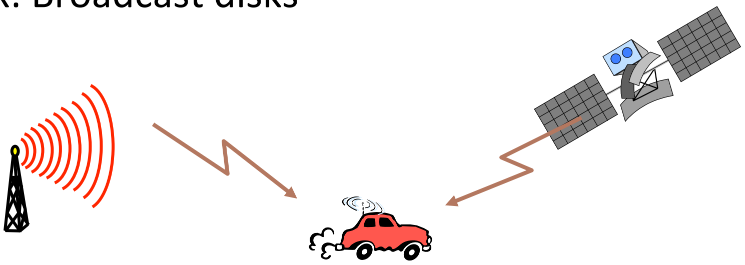
Peer-to-peer replication



- Aka multi-master replication
- Any node in overlay network can propagate updates to other nodes
- Advantage: nodes do not need access to server to synchronize data
- Disadvantage:
  - More complex model (no master copy of data)
  - Lack of knowledge about replication topology
  - Decentralized conflict handling

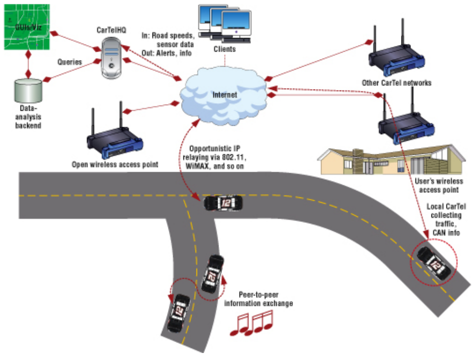
Publish-subscribe

- Publisher emits snippets of data on channels
- Ex: Cellular providers broadcast news items
- Ex: Broadcast disks



Related

- Sensor networks
- Delay-tolerant networking
  - Physical device movement is part of routing
- Infostations
  - Staging area for send/ receiving data


GUIs/Viz
CarTelHQ
In: Road speeds, sensor data
Out: Alerts, info
Clients
Queries
Data-analysis backend
Internet
Other CarTel networks
Open wireless access point
Opportunistic IP relaying via 802.11, WiMAX, and so on
User's wireless access point
Local CarTel collecting traffic, CAN info
Peer-to-peer information exchange

Replication Requirements

	Remote Access	Device Master	Peer to Peer	Publish Subscribe
Continuous Connectivity	✔✔			✔
Update Anywhere		✔✔	✔✔	
Consistency		✔✔	✔✔	✔✔
Topology Independence			✔✔	
Conflict Handling		✔✔	✔✔	
Partial Replication		✔	✔	✔✔

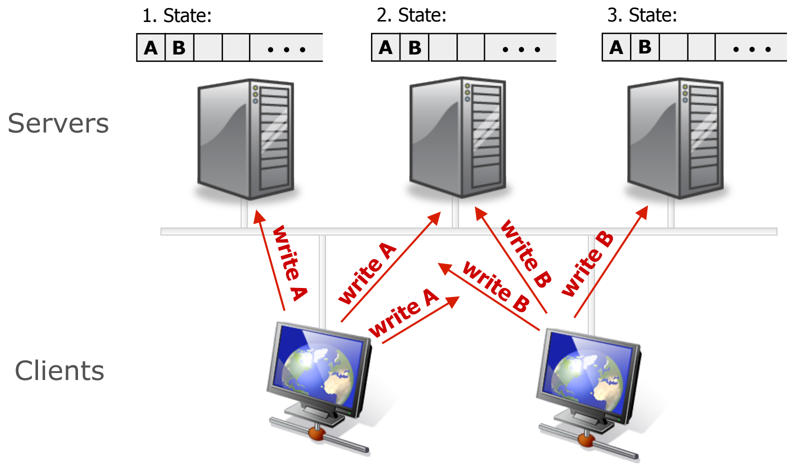
REPLICATED DATA PROTOCOLS

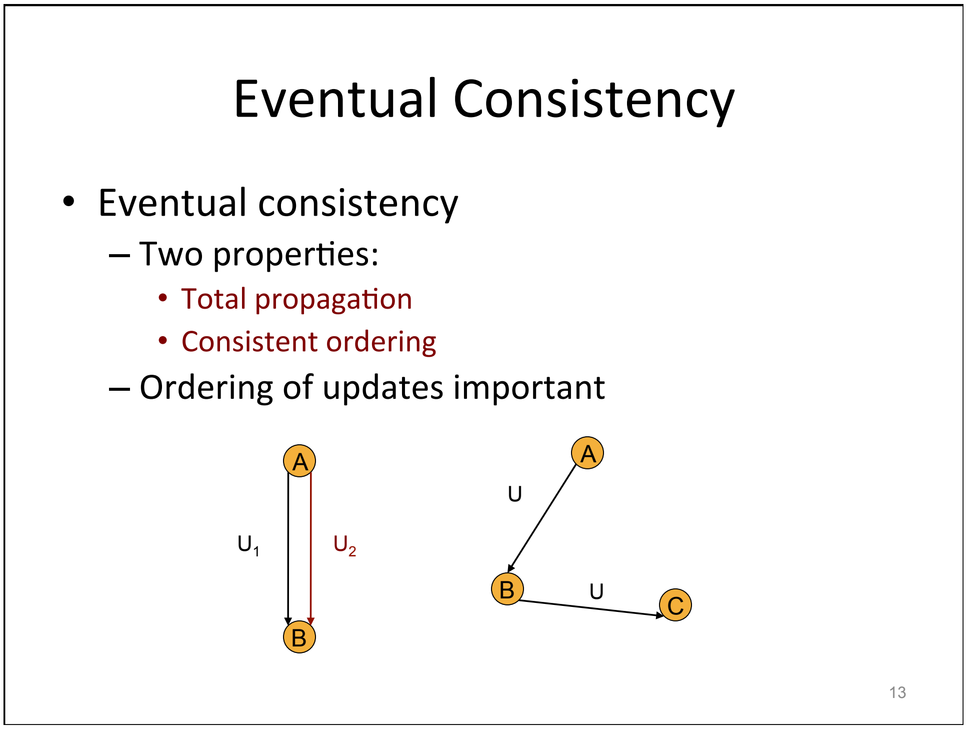
Questions for Data Replication

- What consistency requirements do we guarantee for replicated data?
- How do we represent the updates?
- How do we send updates?
- How do we order the updates?

Data Consistency

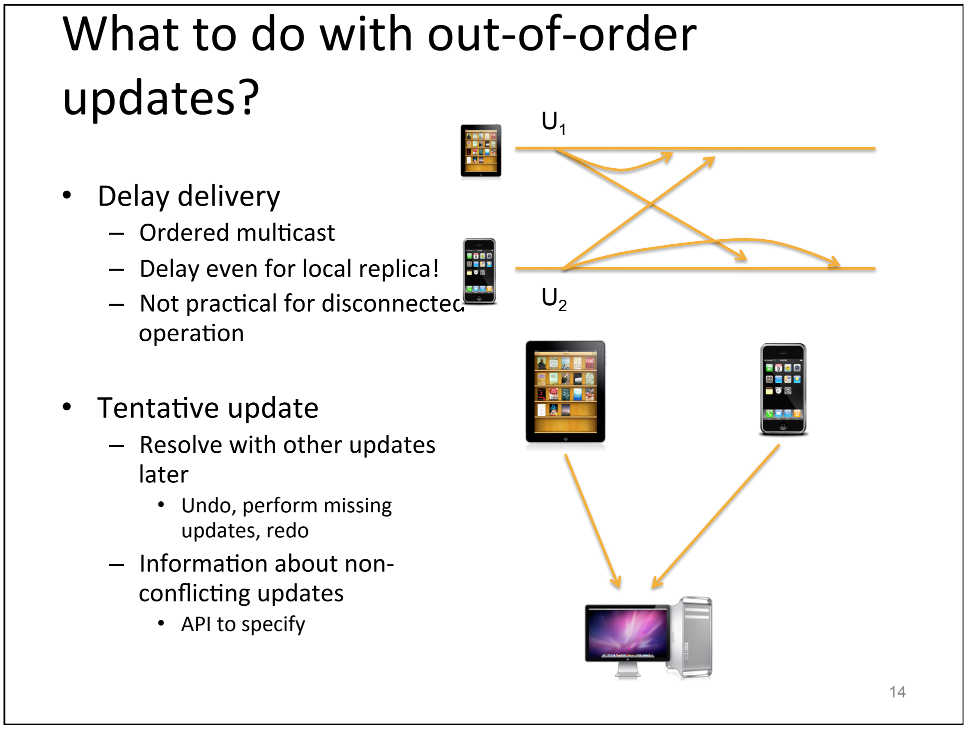
- Strong consistency





Eventual Consistency

- Eventual consistency
  - Two properties:
    - Total propagation
    - Consistent ordering
  - Ordering of updates important



What to do with out-of-order updates?

- Delay delivery
  - Ordered multicast
  - Delay even for local replica!
  - Not practical for disconnected operation
- Tentative update
  - Resolve with other updates later
    - Undo, perform missing updates, redo
  - Information about non-conflicting updates
    - API to specify

Questions for Data Replication

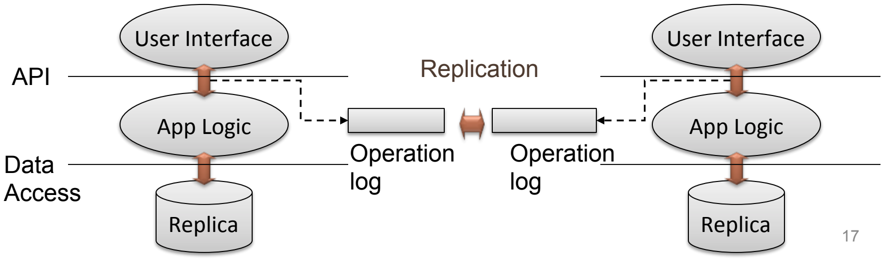
- What consistency requirements do we guarantee for replicated data?
- How do we represent the updates?
- How do we send updates?
- How do we order the updates?

Representing Updates

- Operation-sending protocols
  - Ex: File system updates in CODA
- Item-sending protocols
  - Ex: Disk block updates

Recording Updates (1)

- Log-based systems
  - Replicas will converge to consistent state provided:
    - Each replica receives and applies all updates operations
    - Non-commutative operations applied in same order
    - Operations have deterministic execution
    - Ex: CODA, Bayou

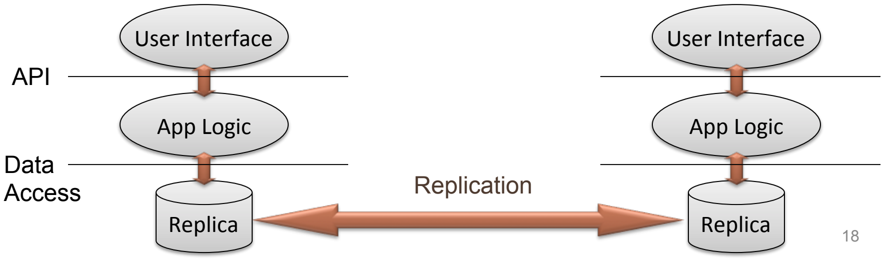


17

Replica = file system
Data item = file

Recording Updates (2)

- State-based systems
  - Attach metadata to items (modified bit, update timestamp, version number)
  - Deleted items: create-delete ambiguity
    - “deleted” bit (tombstone, death cert)
  - Cannot take advantage of operation semantics e.g. to ensure atomicity in update replay

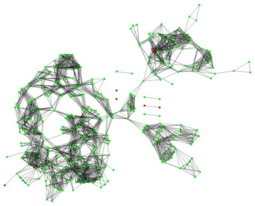


18

Questions for Data Replication

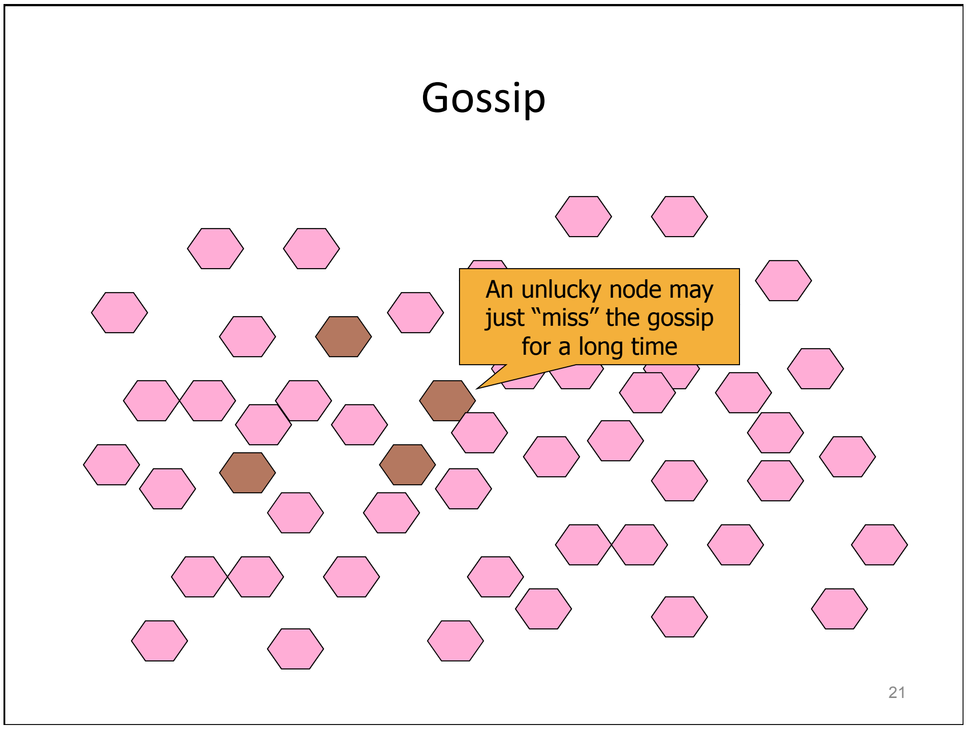
- What consistency requirements do we guarantee for replicated data?
- How do we represent the updates?
- How do we send updates?
- How do we order the updates?

Gossip



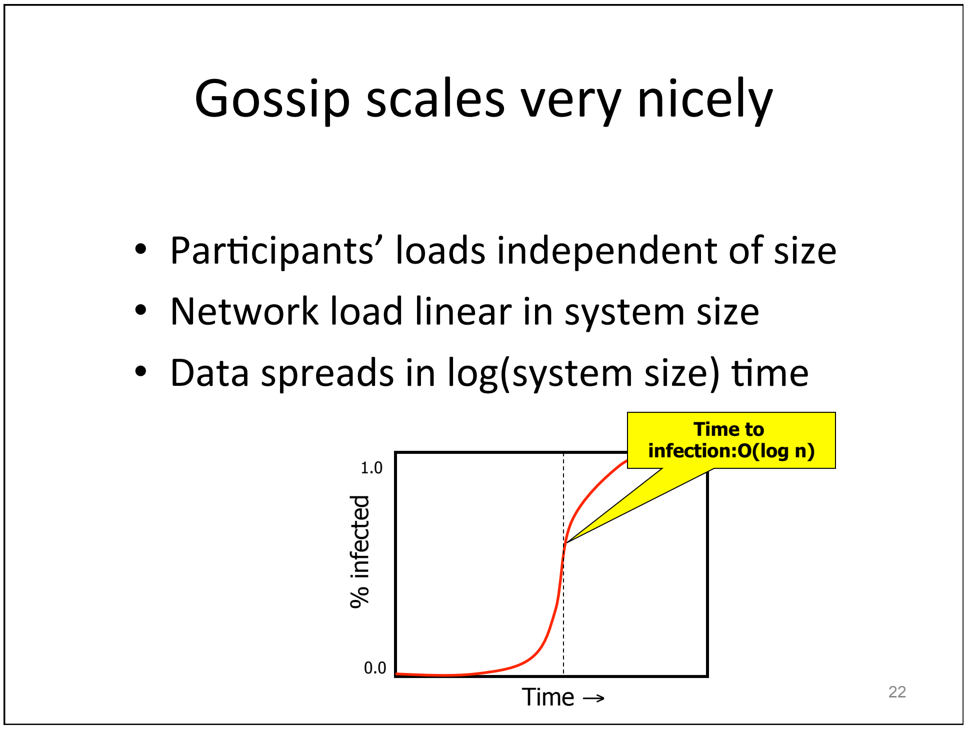
- [t=0] Suppose I know something
- [t=1] I pick you… Now two of us know it.
- [t=2] We each pick … now 4 know it…

- Information spread: exponential rate.
  - Due to re-infection (gossip to an infected node) spreads as 1.8^k after *k* rounds
  - But in O(log(N)) time, N nodes are infected



Gossip

An unlucky node may just “miss” the gossip for a long time



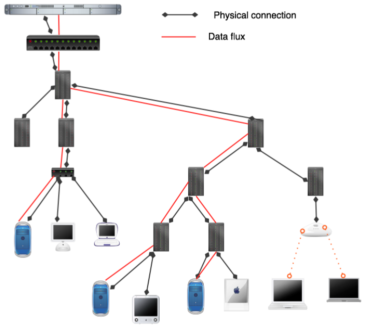
Gossip scales very nicely

- Participants’ loads independent of size
- Network load linear in system size
- Data spreads in log(system size) time

Time to infection:O(log n)

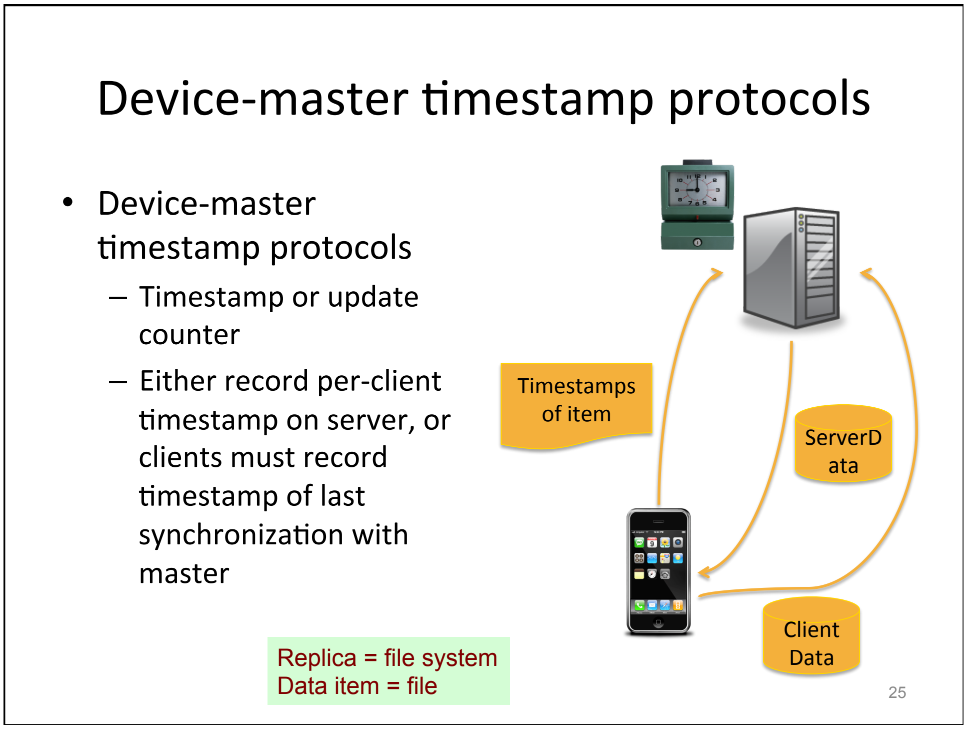
Message Queue Protocols

- Message queue protocols
  - Multicast tree for routing
  - Must notify sender if delivery fails



Modified bit protocol

- Modified bit protocol
  - Simple but only pairwise e.g. hot-syncing PDA



Device-master timestamp protocols

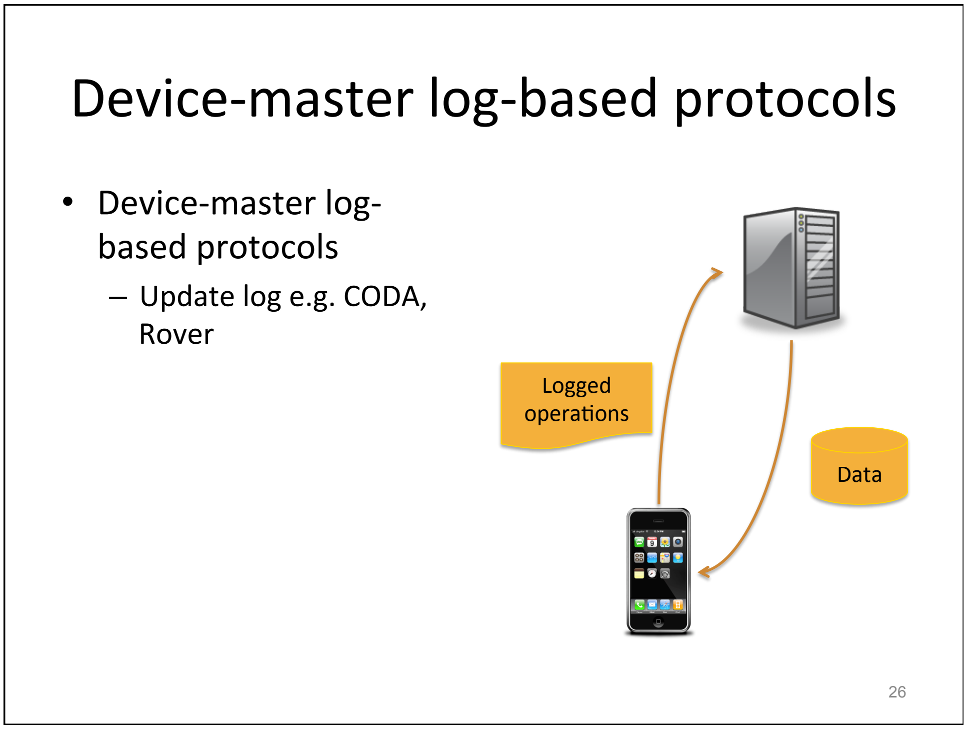
- Device-master timestamp protocols
  - Timestamp or update counter
  - Either record per-client timestamp on server, or clients must record timestamp of last synchronization with master

Timestamps of item

ServerD ata

Client Data

Replica = file system
Data item = file

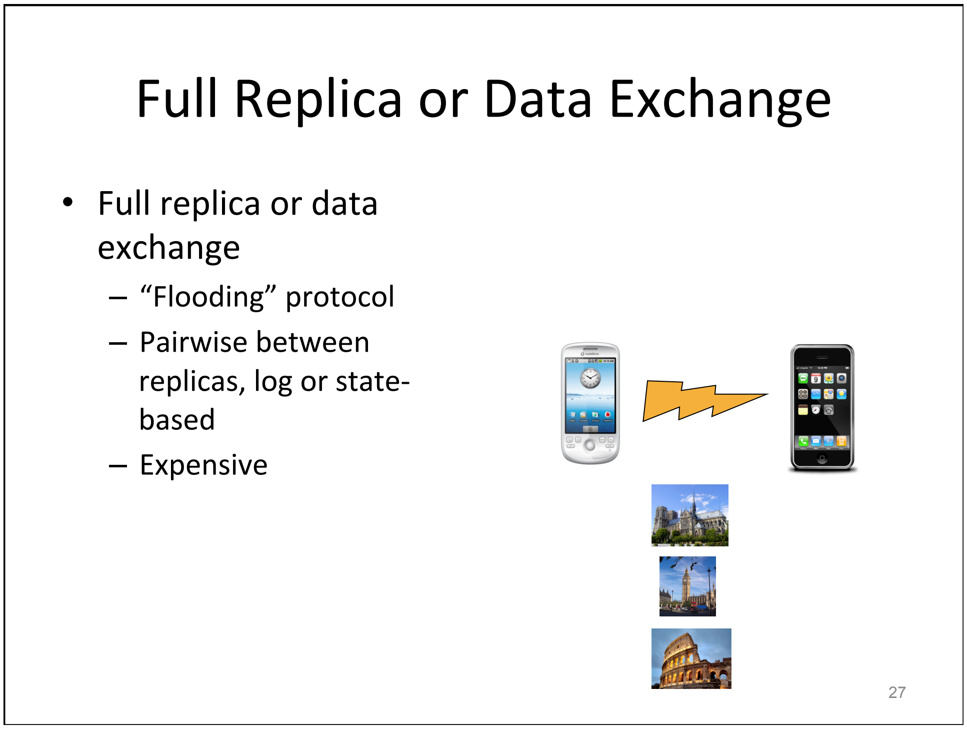


Device-master log-based protocols

- Device-master log-based protocols
  - Update log e.g. CODA, Rover

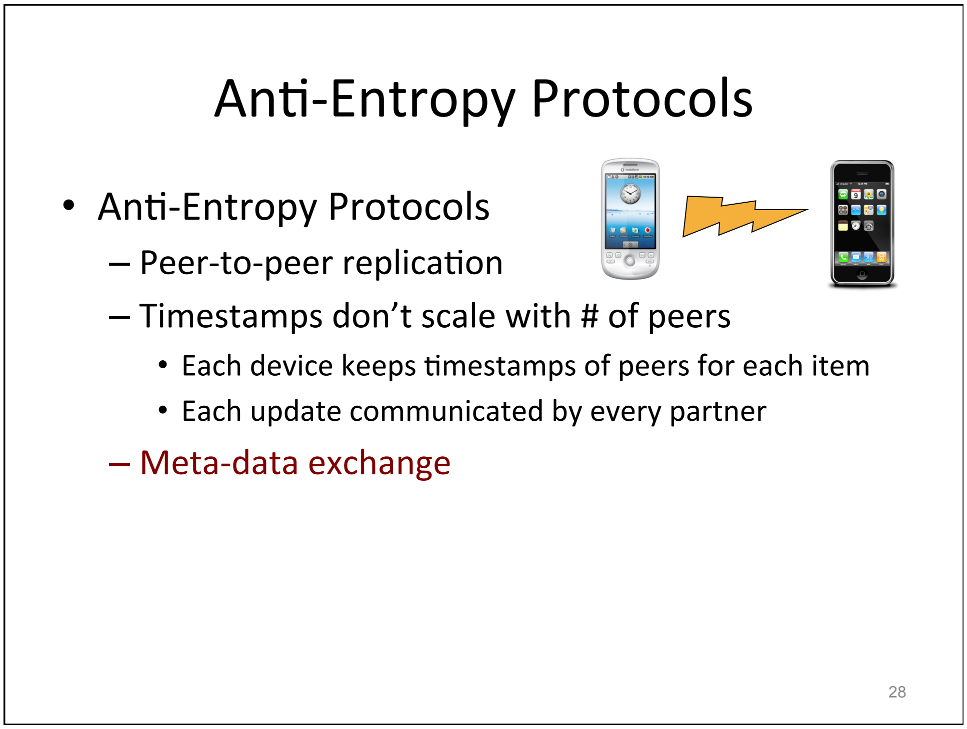
Logged operations

Data



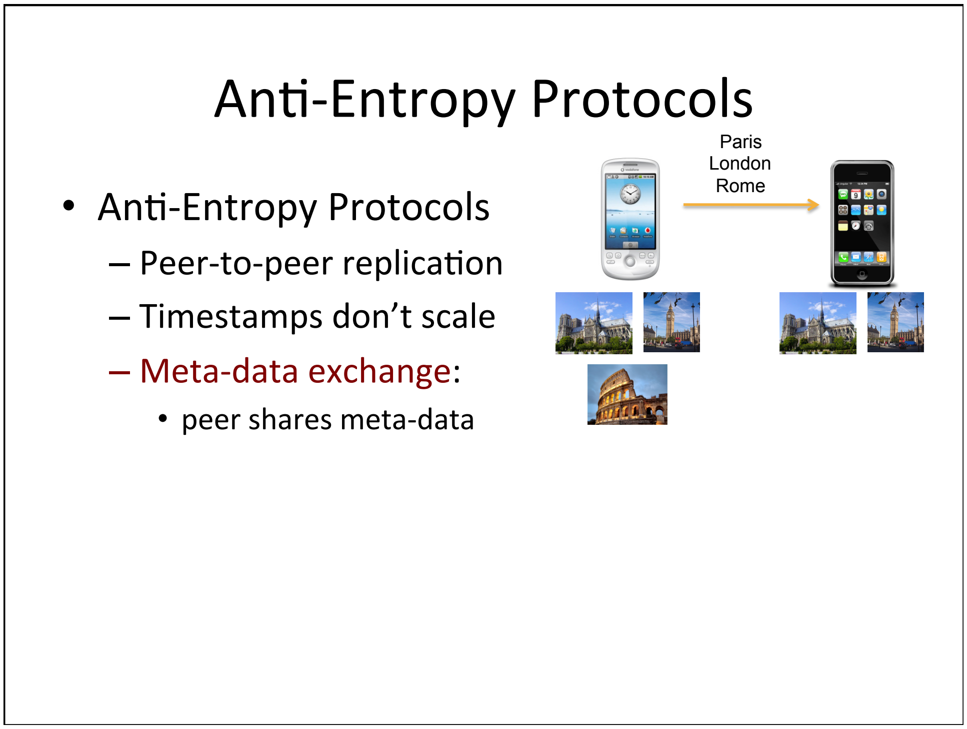
Full Replica or Data Exchange

- Full replica or data exchange
  - “Flooding” protocol
  - Pairwise between replicas, log or state-based
  - Expensive



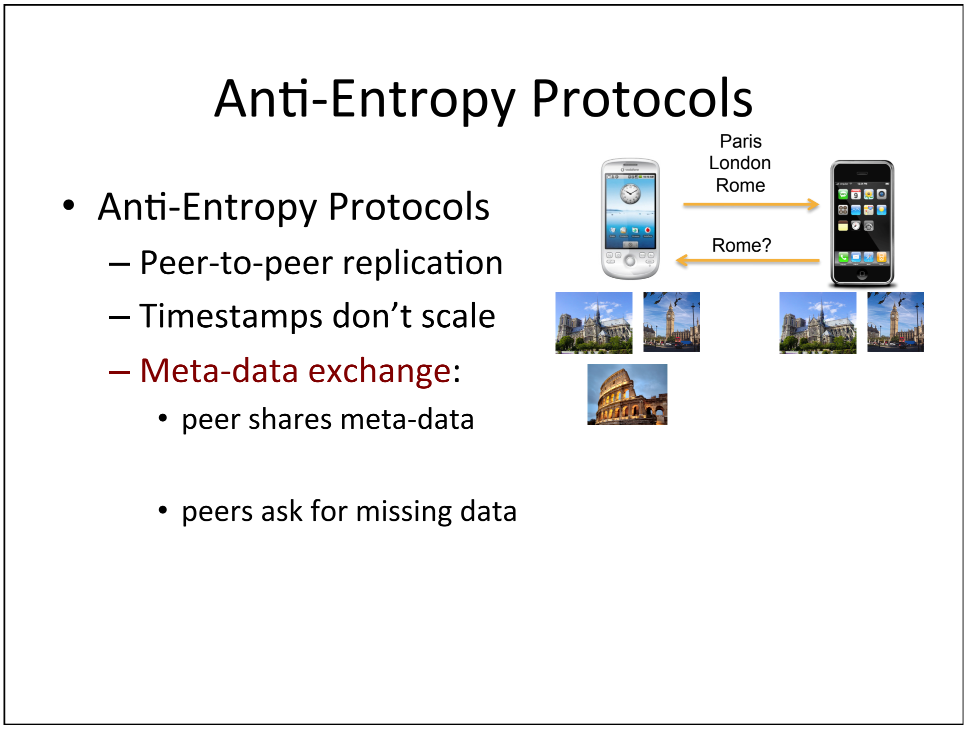
Anti-Entropy Protocols

- Anti-Entropy Protocols
  - Peer-to-peer replication
  - Timestamps don’t scale with # of peers
    - Each device keeps timestamps of peers for each item
    - Each update communicated by every partner
  - Meta-data exchange



Anti-Entropy Protocols

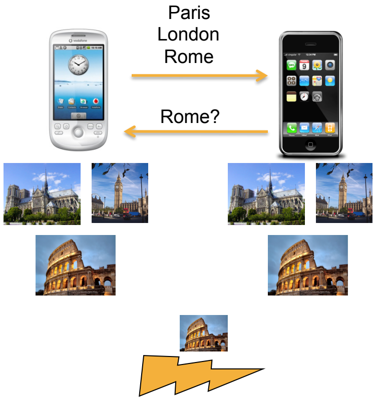
- Anti-Entropy Protocols
  - Peer-to-peer replication
  - Timestamps don’t scale
  - Meta-data exchange:
    - peer shares meta-data



Anti-Entropy Protocols

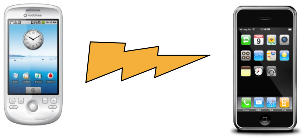
- Anti-Entropy Protocols
  - Peer-to-peer replication
  - Timestamps don’t scale
  - Meta-data exchange:
    - peer shares meta-data
    - peers ask for missing data

Anti-Entropy Protocols

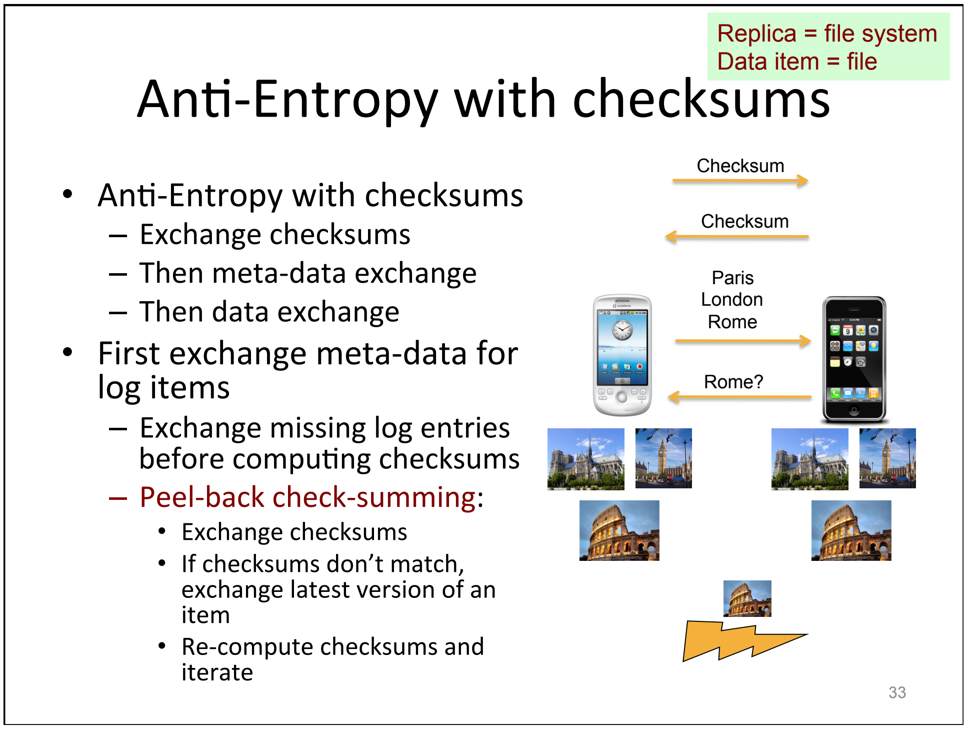


- Anti-Entropy Protocols
  - Peer-to-peer replication
  - Timestamps don’t scale
  - Meta-data exchange:
    - peer shares meta-data
    - peers ask for missing data

Anti-Entropy Protocols



- Anti-Entropy Protocols
  - Peer-to-peer replication
  - Timestamps don’t scale with # of peers
  - Meta-data exchange:
    - peer shares meta-data for replica
    - including version numbers
    - then data exchange
  - Each update received exactly once
  - Expensive



Replica = file system
Data item = file

Anti-Entropy with checksums

- Anti-Entropy with checksums
  - Exchange checksums
  - Then meta-data exchange
  - Then data exchange
- First exchange meta-data for log items
  - Exchange missing log entries before computing checksums
  - Peel-back check-summing:
    - Exchange checksums
    - If checksums don't match, exchange latest version of an item
    - Re-compute checksums and iterate

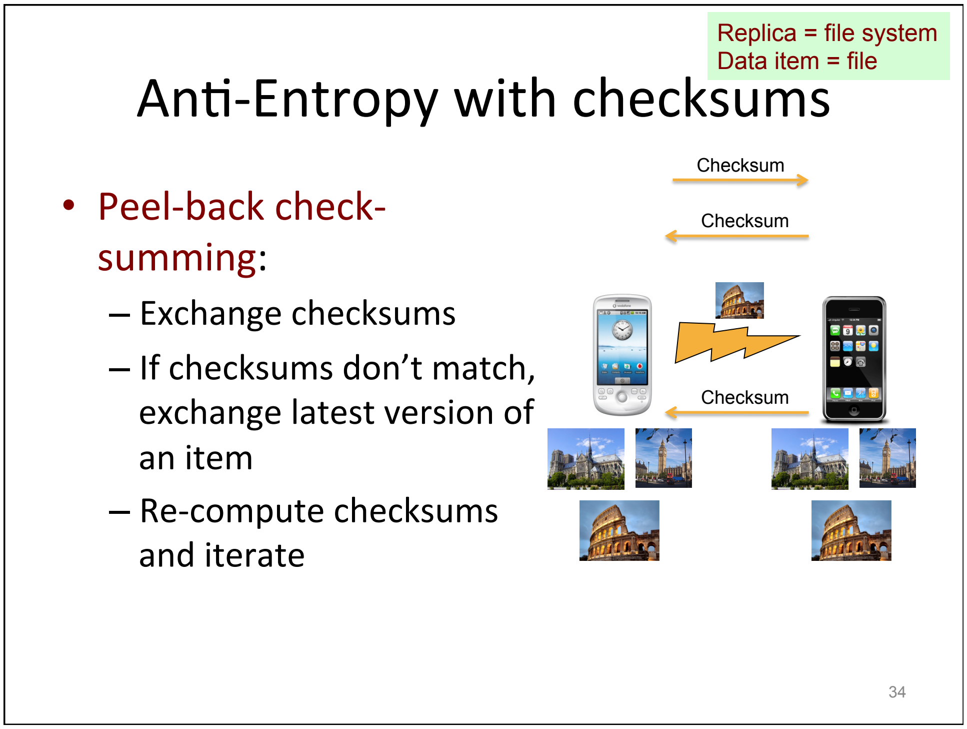
Checksum

Checksum

Paris
London
Rome

Rome?

33



Replica = file system
Data item = file

Anti-Entropy with checksums

- Peel-back check-summing:
  - Exchange checksums
  - If checksums don't match, exchange latest version of an item
  - Re-compute checksums and iterate

Checksum

Checksum

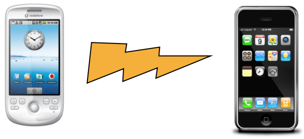
Checksum

34

Knowledge-driven log-based protocols

- Assume each operation uniquely identified
  - “knowledge” of a device: operations it knows about
  - synchronize by sending knowledge
  - store meta-data per replica rather than per data item

Replica = file system
Data item = file



Source

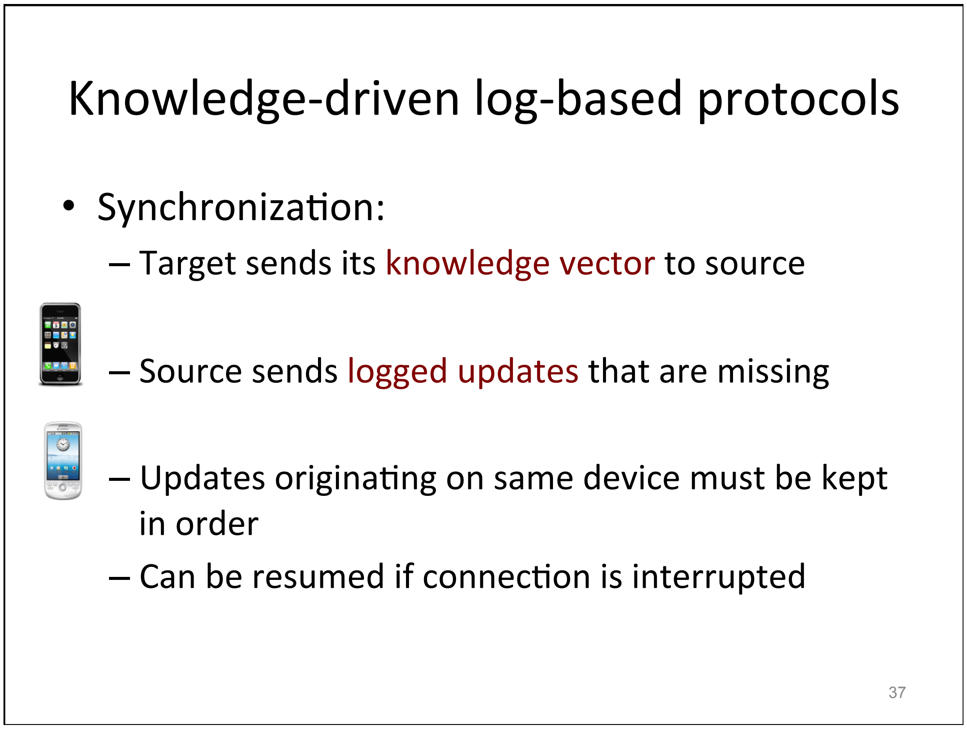
Target

Knowledge-driven log-based protocols

- Accept-timestamp = (replica id, update counter)
  - Uniquely identifies each operation
- Knowledge vector: set of accept-timestamps
  - One entry per replica
  - Assume updates received in order they originated

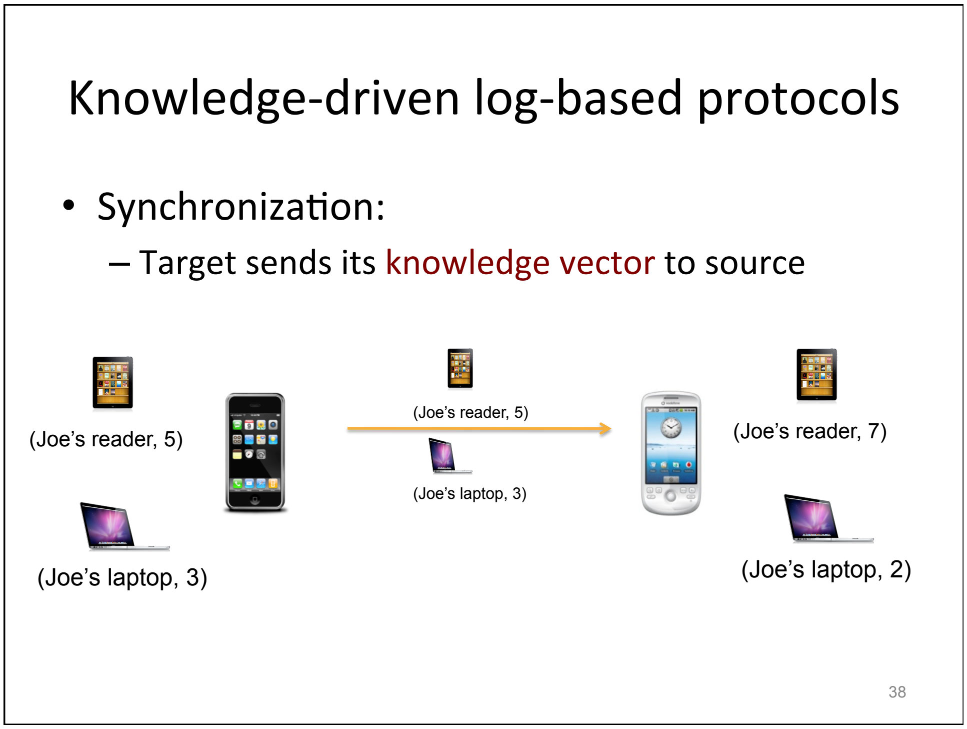
(Joe’s reader, 5)

(Joe’s laptop, 17)



Knowledge-driven log-based protocols

- Synchronization:
  - Target sends its knowledge vector to source
  - Source sends logged updates that are missing
  - Updates originating on same device must be kept in order
  - Can be resumed if connection is interrupted

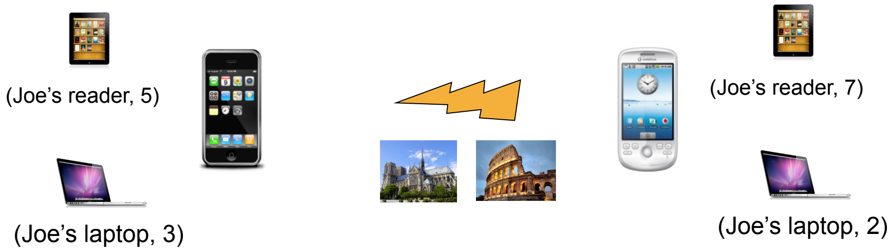


Knowledge-driven log-based protocols

- Synchronization:
  - Target sends its knowledge vector to source

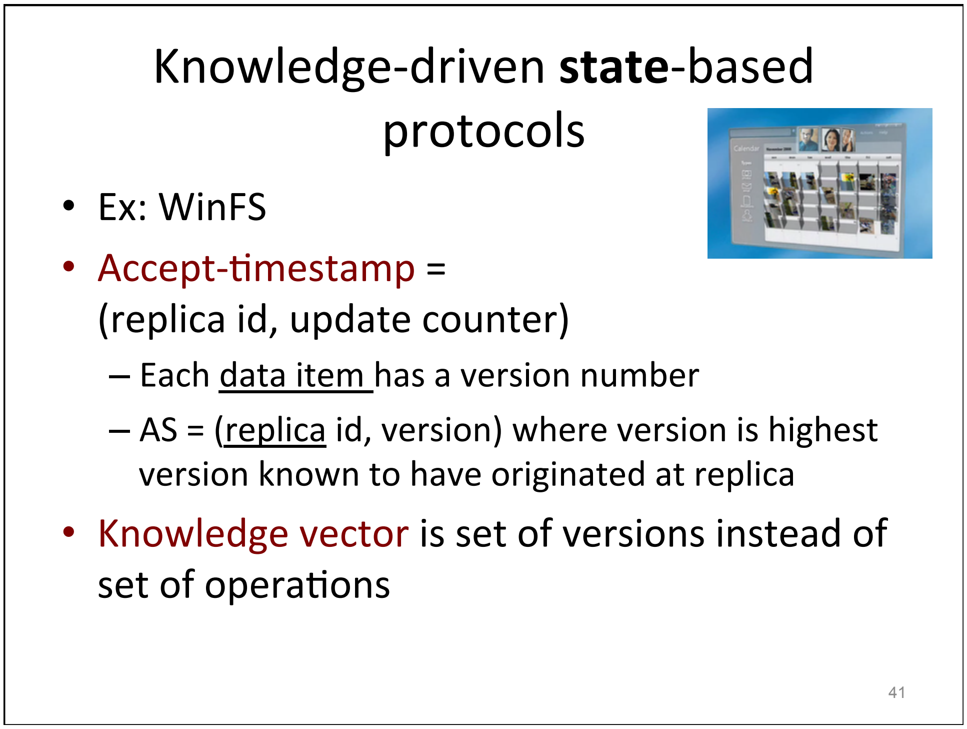
Knowledge-driven log-based protocols

- Synchronization:
  - Source sends logged updates that are missing



Knowledge-driven log-based protocols

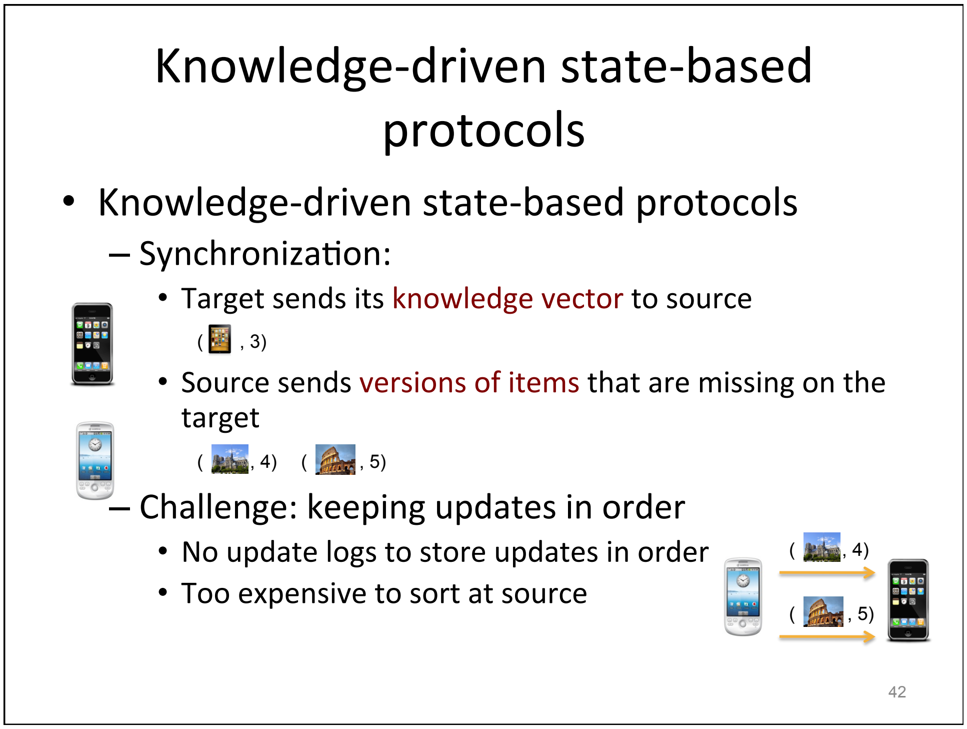
- Freeing up logs
  - Global protocol
  - Or discard log prefixes, keep omitted vector
    - Vector of knowledge for discarded logs
    - Fail over to metadata-driven protocol if necessary
      - If target knowledge omits updates pruned from source logs



Knowledge-driven **state**-based protocols

- Ex: WinFS
- Accept-timestamp = (replica id, update counter)
  - Each data item has a version number
  - AS = (replica id, version) where version is highest version known to have originated at replica
- Knowledge vector is set of versions instead of set of operations

41



Knowledge-driven state-based protocols

- Knowledge-driven state-based protocols
  - Synchronization:
    - Target sends its knowledge vector to source

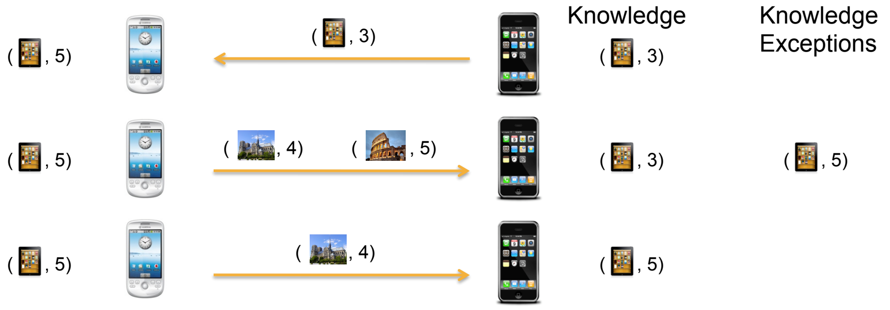
      (, 3)
    - Source sends versions of items that are missing on the target

      (, 4) (, 5)
  - Challenge: keeping updates in order
    - No update logs to store updates in order
    - Too expensive to sort at source

42

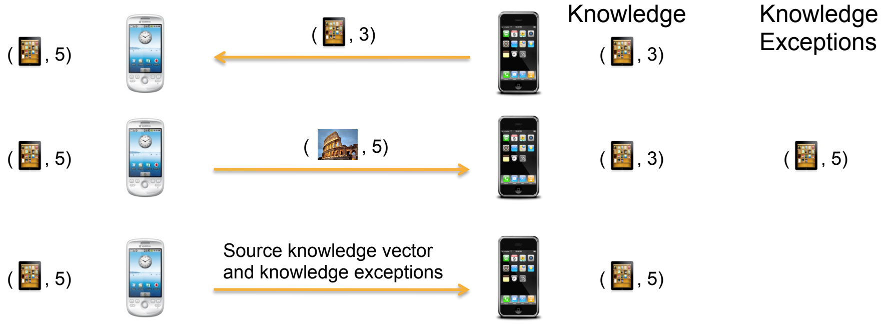
Knowledge-driven state-based protocols

- Knowledge-driven state-based protocols
  - Knowledge Exception: for out of order updates



Knowledge-driven state-based protocols

- Knowledge-driven state-based protocols
  - Learned knowledge: to remove knowledge exns

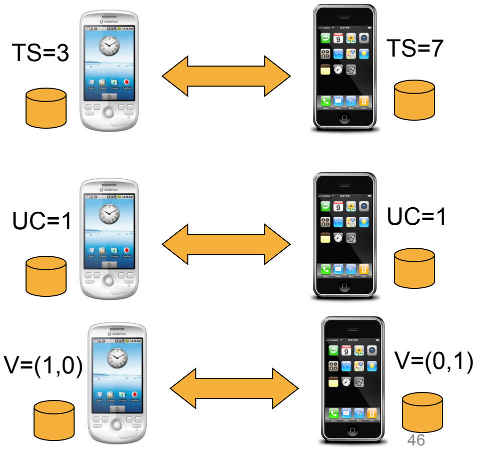


Questions for Data Replication

- What consistency requirements do we guarantee for replicated data?
- How do we represent the updates?
- How do we send updates?
- How do we order the updates?

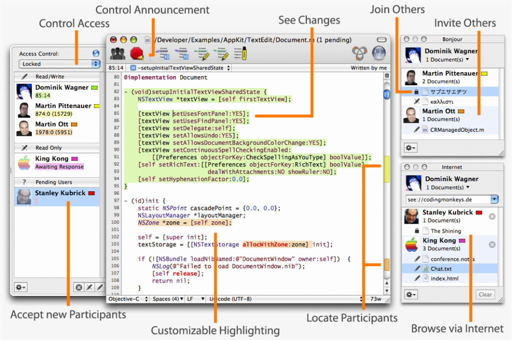
Ordering Updates (1)

- Ordered delivery
- Sequencers
  - Or Paxos algorithm
- Update timestamps
  - Logical timestamps ensure causal order
- Update counters
  - Concurrent updates
  - Resolve with device id
- Version vectors (like vector clocks)



Ordering Updates (2)

- Operation transformation
  - Used e.g. for concurrent editing
  - For n operations, need n^2 transformations

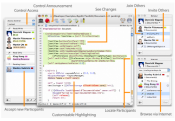
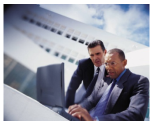


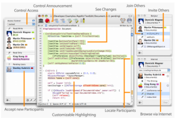
```
@implementation Document

- (void)setupInitialTextViewSharedState {
    NSTextView *textView = [self firstTextView];

    [textView setUsesFontPanel:YES];
    [textView setUsesFindPanel:YES];
    [textView setDelegate:self];
    [textView setAllowsUndo:YES];
    [textView setAllowsDocumentBackgroundColorChange:YES];
    [textView setContinuousSpellCheckingEnabled:
        [[Preferences objectForKey:CheckSpellingAsYouType] boolValue]];
    [self setRichText:[[Preferences objectForKey:RichText] boolValue]
                    dealWithAttachments:NO showRuler:NO];
    [self setHyphenationFactor:0.0];
}

- (id)init {
    static NSPoint cascadePoint = {0.0, 0.0};
    NSLayoutManager *layoutManager;
    NSZone *zone = [self zone];

    self = [super init];
    textStorage = [[NSTextStorage allocWithZone:zone] init];

    if (![NSBundle loadNibNamed:@"DocumentWindow" owner:self])  {
        NSLog(@"Failed to load DocumentWindow.nib");
        [self release];
        return nil;
    }
```

47

Ordering Updates (2)

- Operation transformation
  - Used e.g. for concurrent editing
  - For n operations, need n^2 transformations

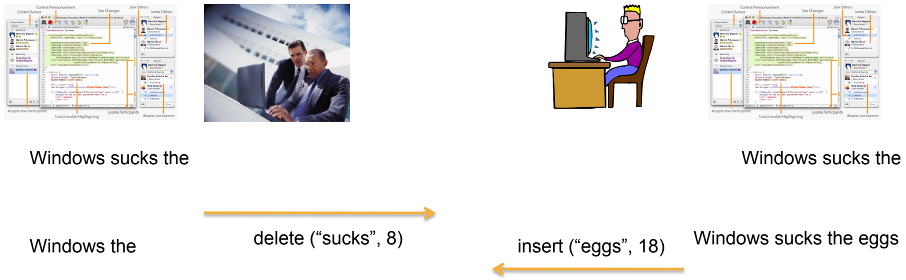


Windows sucks the

Windows sucks the

Ordering Updates (2)

- Operation transformation
  - Used e.g. for concurrent editing
  - For n operations, need n^2 transformations



Windows sucks the

Windows sucks the

Windows the

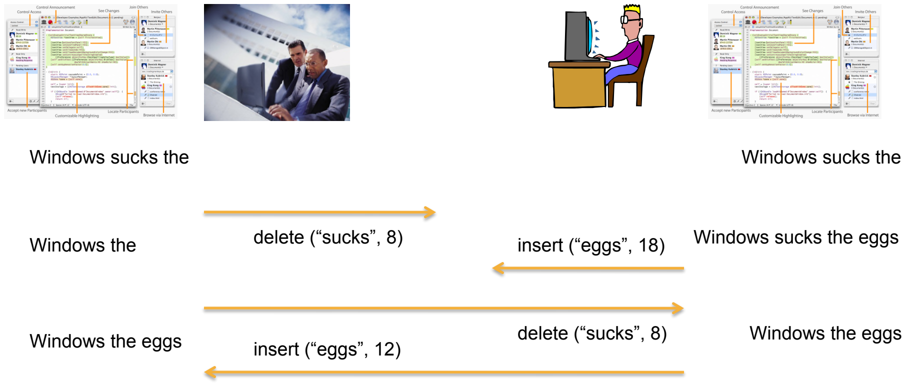
delete (“sucks”, 8)

insert (“eggs”, 18)

Windows sucks the eggs

Ordering Updates (2)

- Operation transformation
  - Used e.g. for concurrent editing
  - For n operations, need n^2 transformations



Windows sucks the

Windows sucks the

Windows the

delete (“sucks”, 8)

insert (“eggs”, 18)

Windows sucks the eggs

Windows the eggs

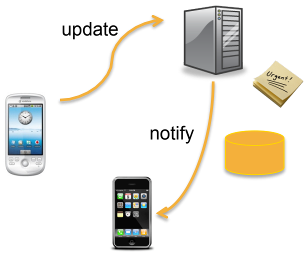
insert (“eggs”, 12)

delete (“sucks”, 8)

Windows the eggs

PARTIAL REPLICATION

Access-based caching



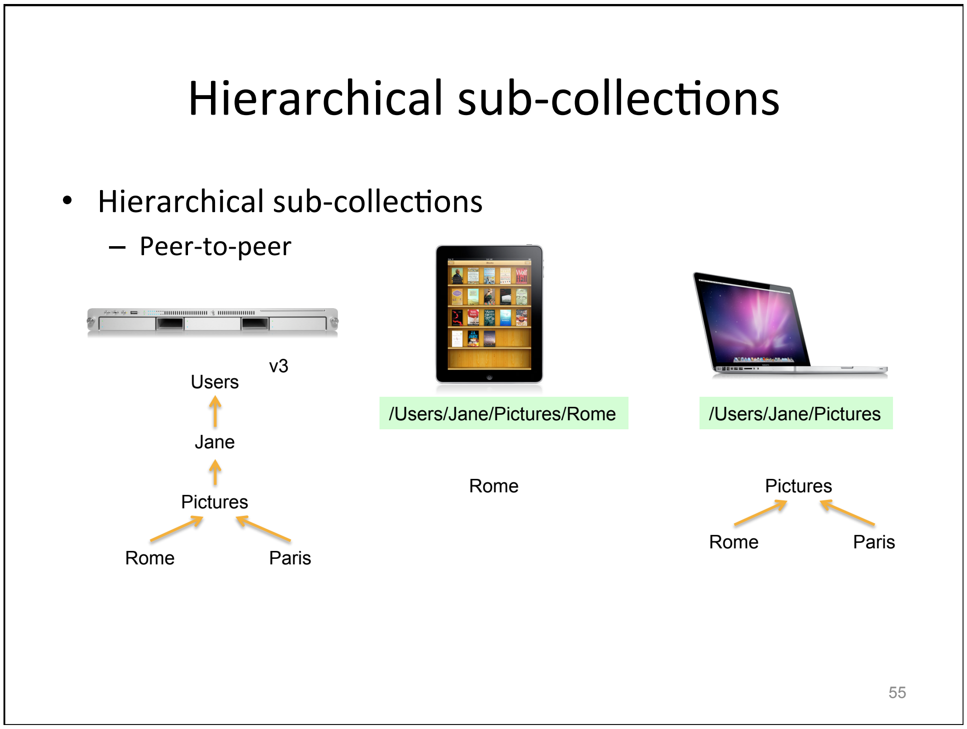
- Access-based caching
  - Problems with callbacks (cf AFS):
    - Callbacks may be missed
    - Server may not know which items are cached
    - Use modified-bit, update timestamp, metadata exchange or knowledge-driven protocols instead
  - Knowledge-driven protocols: devices that are synchronizing are assumed to cache same items
  - Metadata exchange: expensive unless minimize # of items involved
    - sync initiated by caching device

Partial replication

- Policy-based hoarding
  - CODA
- Topic-based channels
  - Disseminate with gossip
  - Peer-to-peer
    - May receive different channels from different sets of peers
    - Knowledge-driven: one knowledge vector per channel

Hierarchical sub-collections

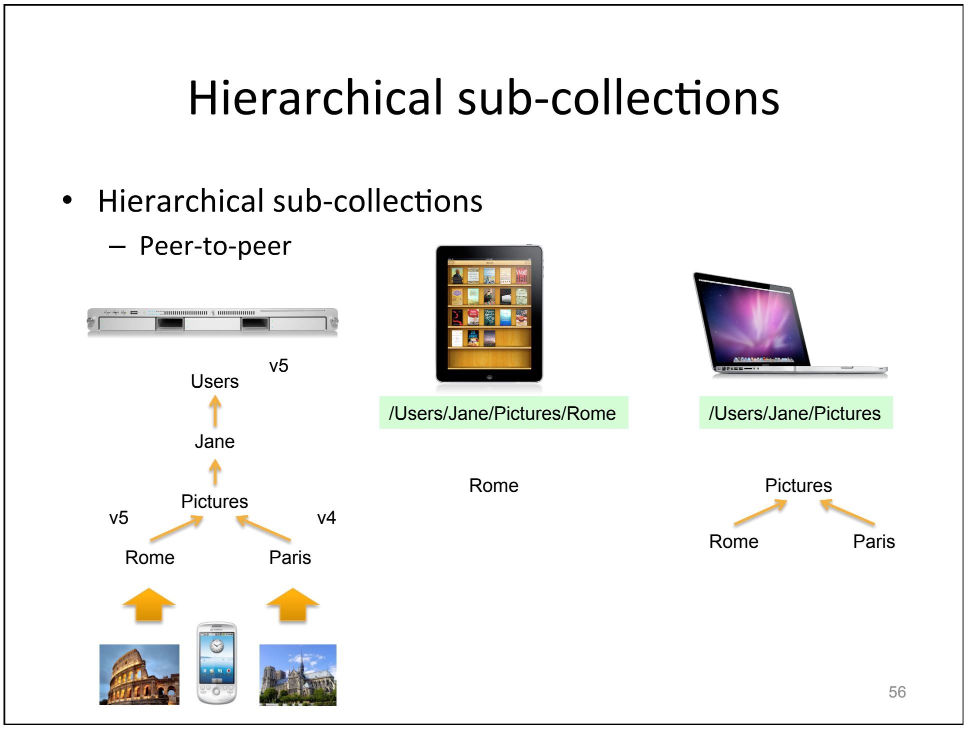
- Hierarchical sub-collections
  - Ex: Mail folders on laptop, inbox only on cell phone
  - Device-master: master only sends updates for sub-collection
  - Peer-to-peer: single knowledge vector will not handle sub-collections



Hierarchical sub-collections

- Hierarchical sub-collections
  - Peer-to-peer

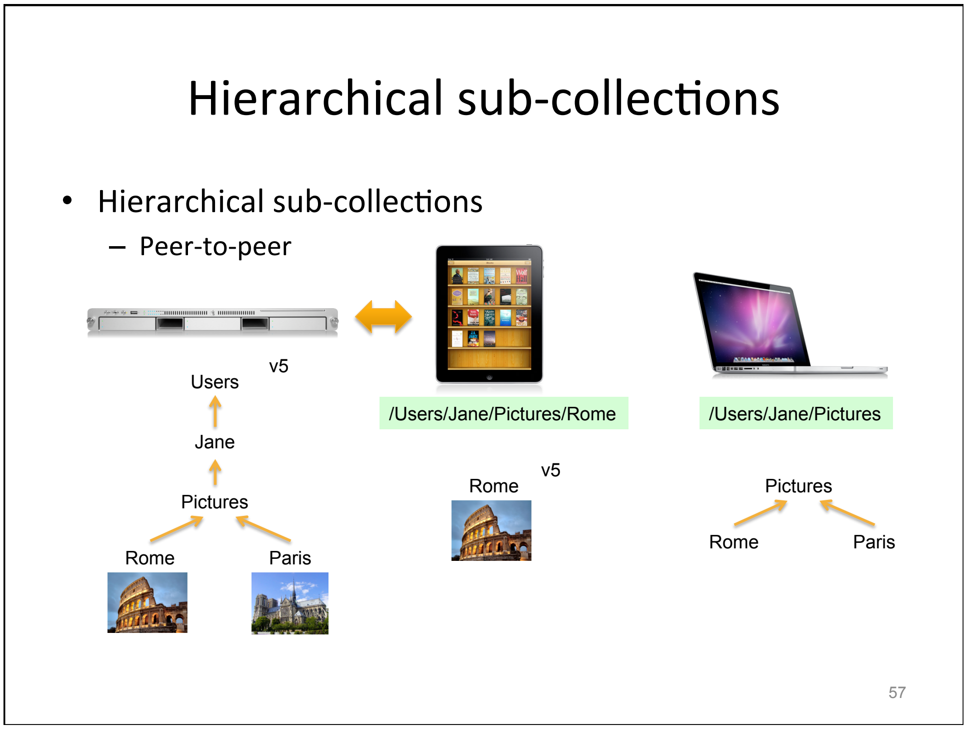
55



Hierarchical sub-collections

- Hierarchical sub-collections
  - Peer-to-peer

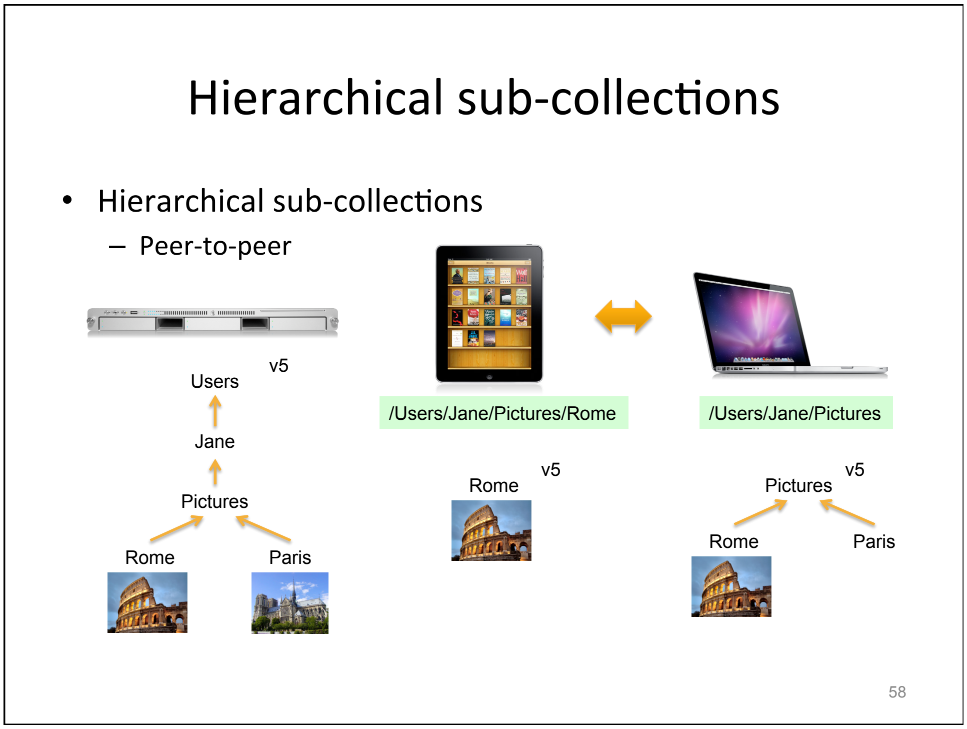
56



Hierarchical sub-collections

- Hierarchical sub-collections
  - Peer-to-peer

57



Hierarchical sub-collections

- Hierarchical sub-collections
  - Peer-to-peer

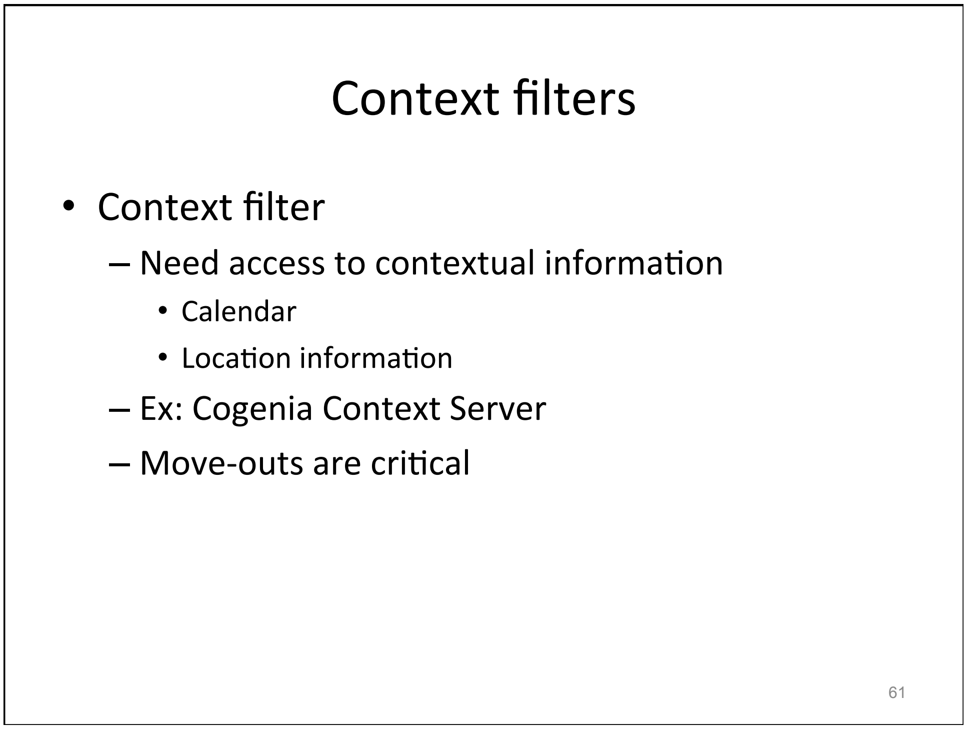
58

Hierarchical sub-collections

- Hierarchical sub-collections
  - Ex: Mail folders on laptop, inbox only on cell phone
  - Device-master: master only sends updates for sub-collection
  - Peer-to-peer: single knowledge vector will not handle sub-collections
    - Separate knowledge vector for each sub-collection
    - Updates may be received more than once

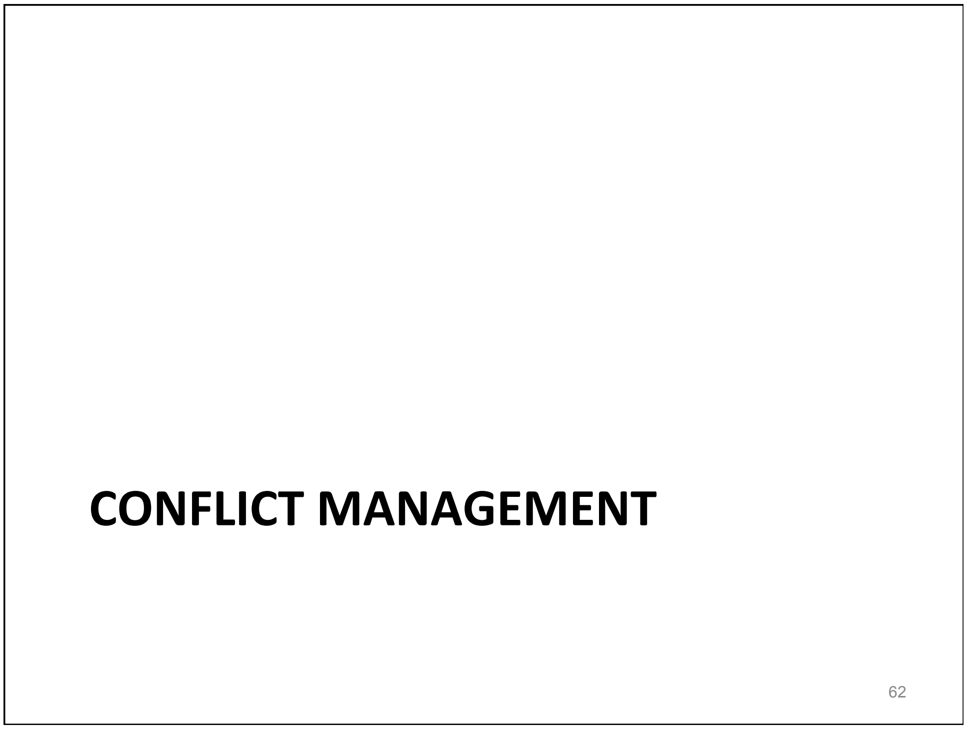
Content filters

- Content filters
  - Filter query should be on sync partners
  - Device-master state-based: easy
  - Log-based: filter based on type or operation or item it updates
  - Move-out: what if cached item is updated and no longer matches filter?
  - Peer-to-peer: topology issue (full-partial-full)
- Metadata exchange supports content filters and move-out
  - open issue for knowledge-driven

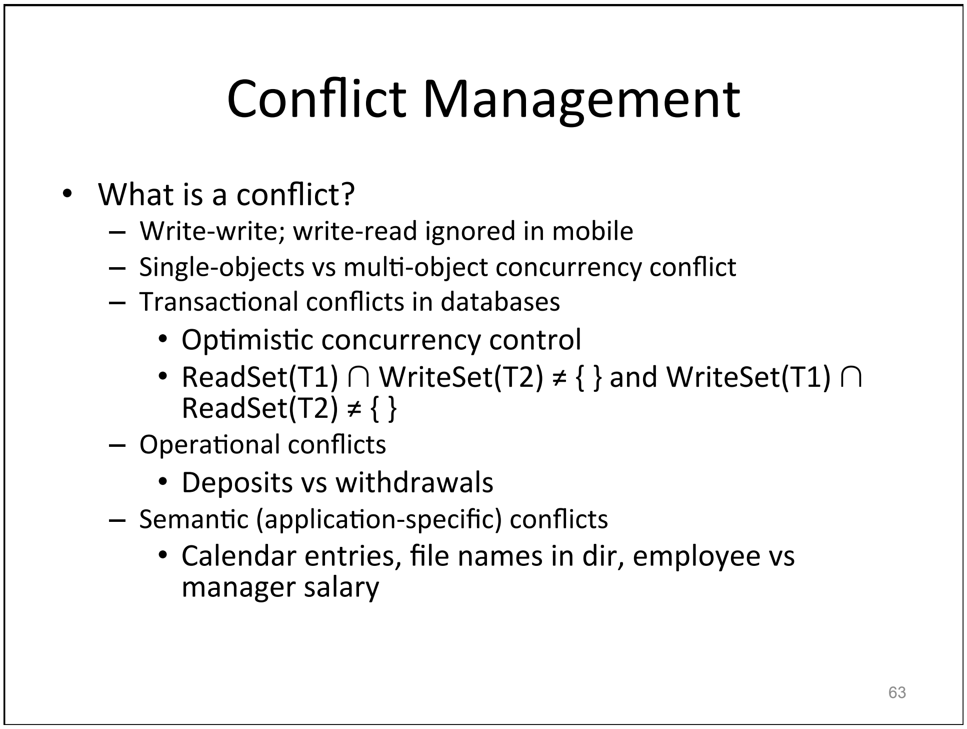


Context filters

- Context filter
  - Need access to contextual information
    - Calendar
    - Location information
  - Ex: Cogenia Context Server
  - Move-outs are critical

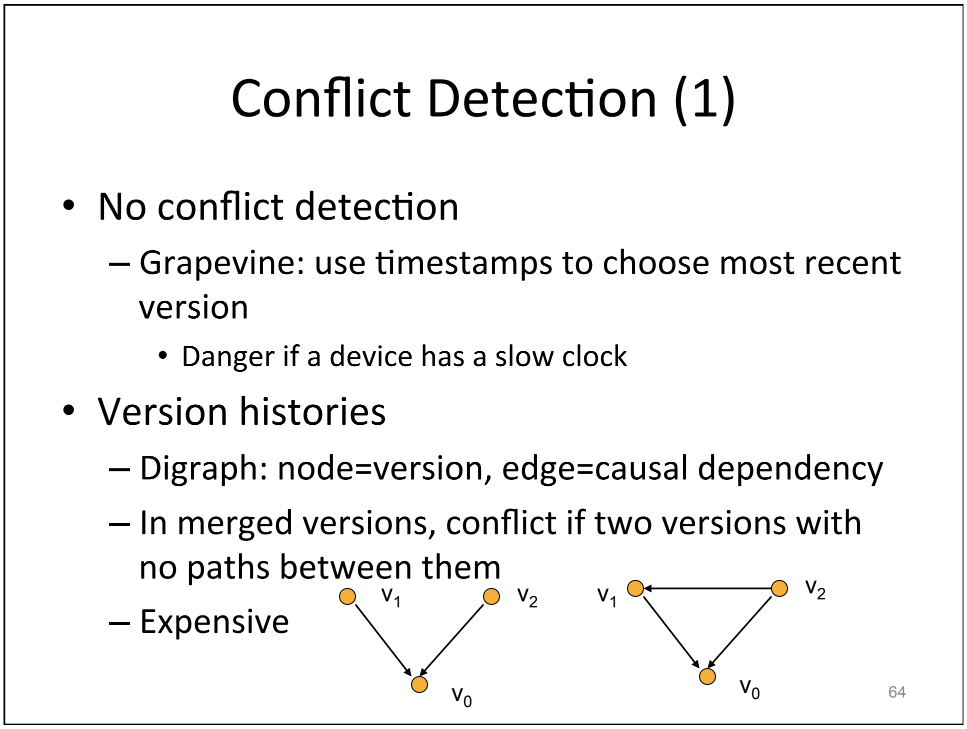


CONFLICT MANAGEMENT



Conflict Management

- What is a conflict?
  - Write-write; write-read ignored in mobile
  - Single-objects vs multi-object concurrency conflict
  - Transactional conflicts in databases
    - Optimistic concurrency control
    - ReadSet(T1) ∩ WriteSet(T2) ≠ { } and WriteSet(T1) ∩ ReadSet(T2) ≠ { }
  - Operational conflicts
    - Deposits vs withdrawals
  - Semantic (application-specific) conflicts
    - Calendar entries, file names in dir, employee vs manager salary



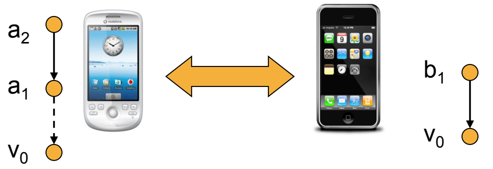
Conflict Detection (1)

- No conflict detection
  - Grapevine: use timestamps to choose most recent version
    - Danger if a device has a slow clock
- Version histories
  - Digraph: node=version, edge=causal dependency
  - In merged versions, conflict if two versions with no paths between them
  - Expensive

v_1
v_2
v_0
v_1
v_2
v_0

Conflict Detection (2)

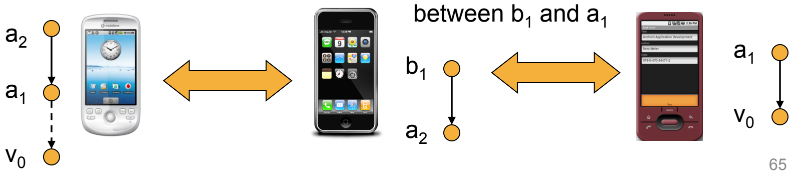
- Previous versions
  - Only store the previous version with an item



Fail to find version conflict

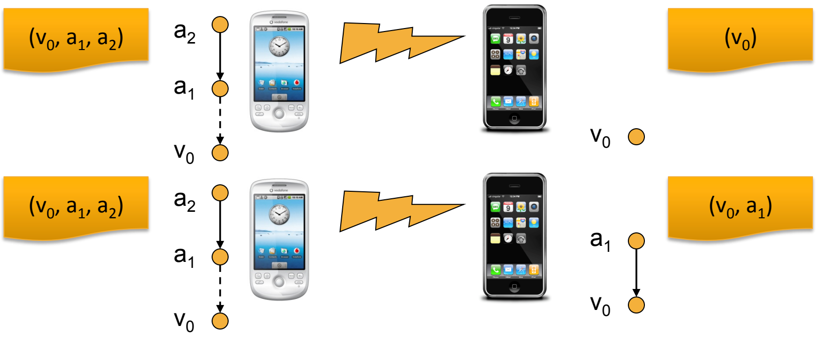
- Suppose conflict if neither version is predecessor of the other?

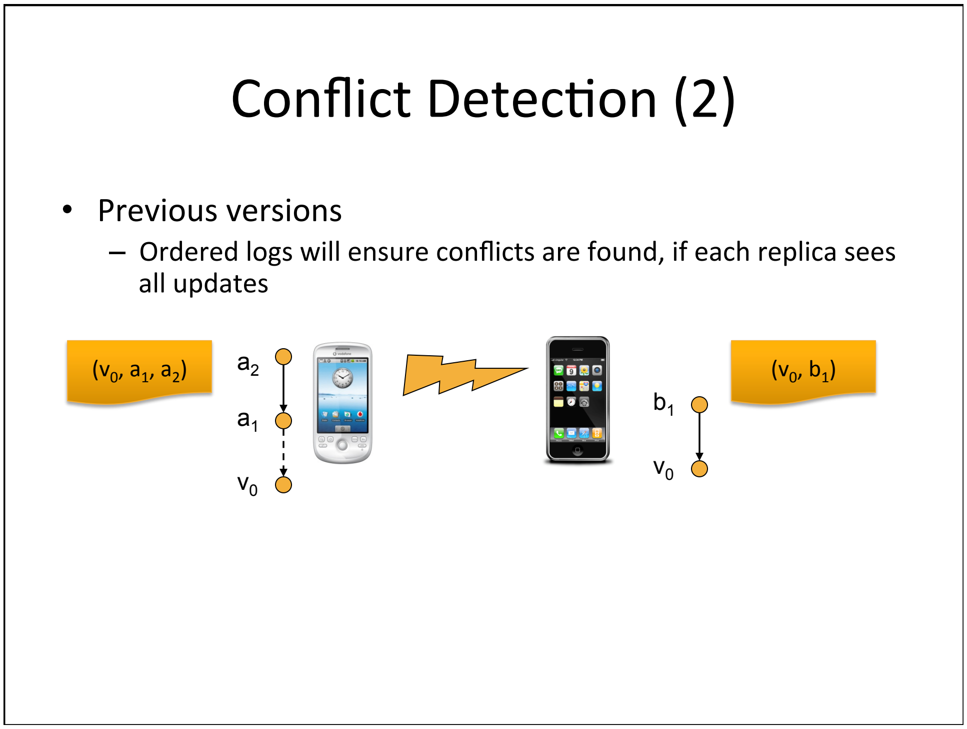
False conflict between b_1 and a_1



Conflict Detection (2)

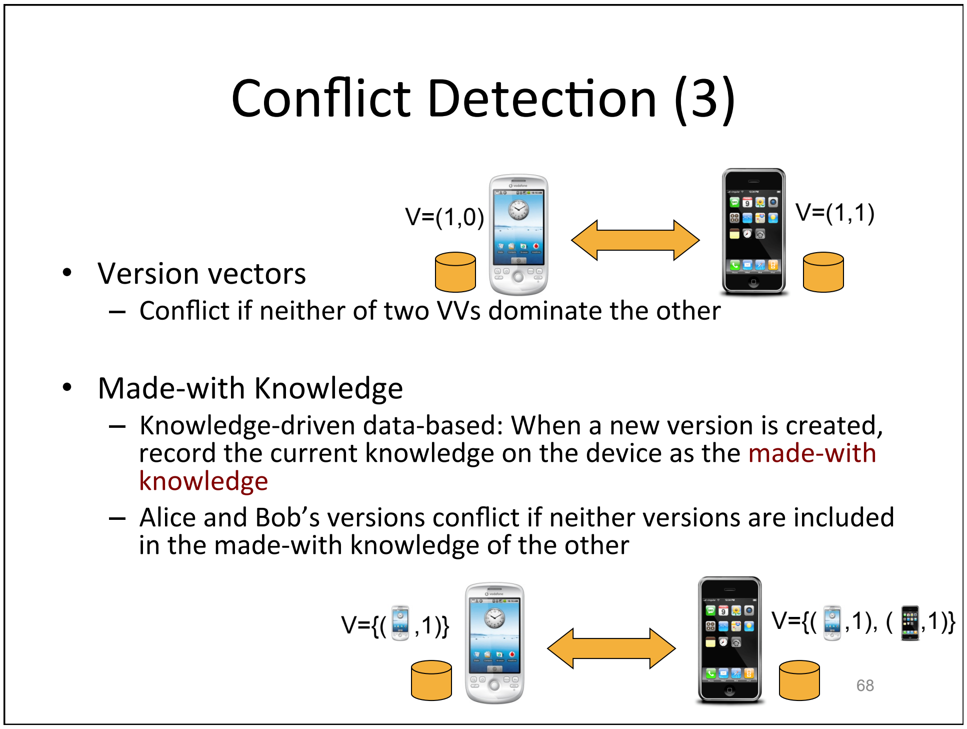
- Previous versions
  - Ordered logs will ensure conflicts are found, if each replica sees all updates





Conflict Detection (2)

- Previous versions
  - Ordered logs will ensure conflicts are found, if each replica sees all updates



Conflict Detection (3)

- Version vectors
  - Conflict if neither of two VVs dominate the other

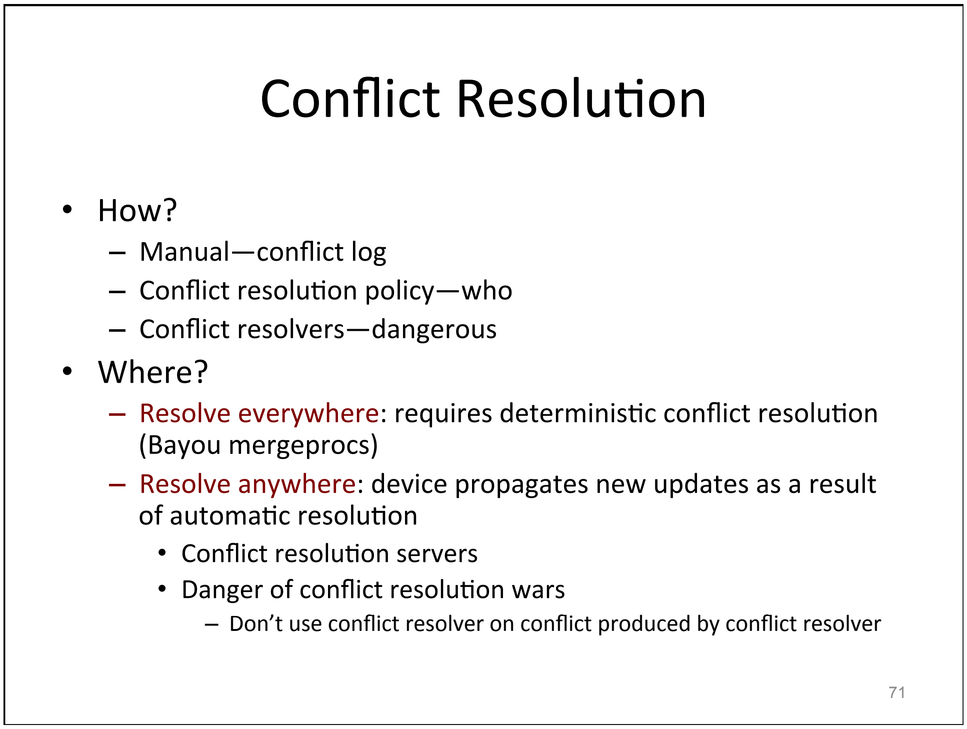
- Made-with Knowledge
  - Knowledge-driven data-based: When a new version is created, record the current knowledge on the device as the made-with knowledge
  - Alice and Bob's versions conflict if neither versions are included in the made-with knowledge of the other

Conflict Detection (4)

- Read-sets
  - – Read-sets for optimistic transactions
  - – When an item is received at device, does its read-set include an item that has a different version on the device?
- Operation conflict tables
  - – Conflict may be based on parameters
  - – Requires infinite log to search for conflicting operations
  - – Impractical
- Integrity constraints
  - – Specify constraints as data invariants
  - – Database triggers for violations

Conflict Detection (5)

- Dependency check
  - – General scheme for implementing conflict detection
  - – For each logged update, store a query and an expected set of results
  - – Examples
    - Previous version
    - Read-set
    - Integrity constraints



Conflict Resolution

- How?
  - Manual—conflict log
  - Conflict resolution policy—who
  - Conflict resolvers—dangerous
- Where?
  - Resolve everywhere: requires deterministic conflict resolution (Bayou mergeprocs)
  - Resolve anywhere: device propagates new updates as a result of automatic resolution
    - Conflict resolution servers
    - Danger of conflict resolution wars
      - Don't use conflict resolver on conflict produced by conflict resolver