ZZFS: A file system for spontaneous users

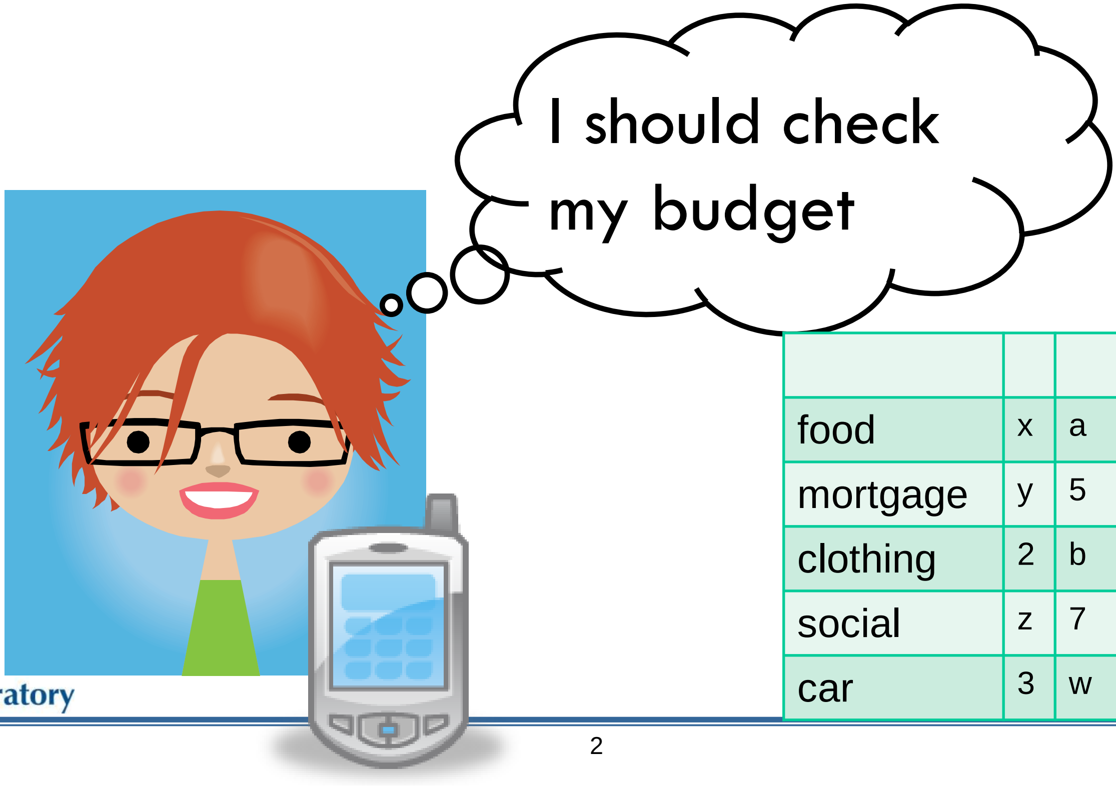
Michelle L. Mazurek[1], Eno Thereska[2], Dinan Gunawardena[2], Richard Harper[2], James Scott[2]

[1]Carnegie Mellon University

[2]Microsoft Research, Cambridge UK

In ideal world ….

- Alice wants to check her financial spreadsheet
- It's available and up-to-date on her phone
  - Even if she updated it from her laptop
  - Even if her husband updated it from his phone



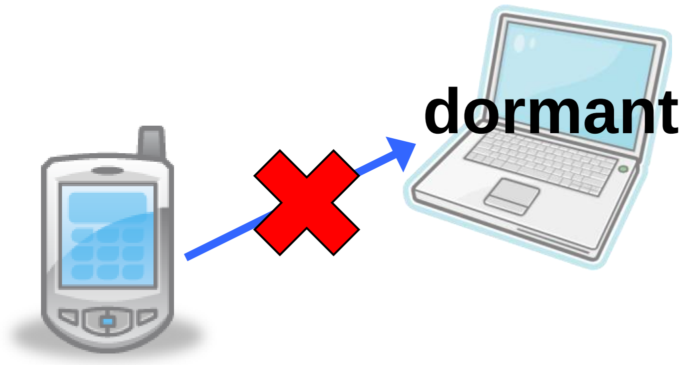
food	x	a	9
mortgage	y	5	13
clothing	2	b	r
social	z	7	4
car	3	w	x

Personal doesn't mean simple

- Requires collaboration
  - Household management (e.g. financials)
  - Beyond the household
- Requires coordinating devices
  - Non-homogenous, frequently changing
- Requires handling writes
  - Example: Listening to music
- **Ideally, all of this should be seamless**

One key problem: Unavailable devices

- Read financials, but laptop is off
- Switch from laptop to tablet
  - Never on at the same time; how to sync?
- Update remote data
  - Could have been pre-fetched if device was on
- **Unavailable devices inhibit on-demand data**



Human factors are important

- Eventual consistency (e.g. Coda)
  - Users want data now
- Planning and hoarding (Perspective, Anzere)
  - Users are spontaneous
- In a public cloud
  - Users want trust, control
- Best of each?
  - Availability and trust/control, without planning

ZZFS: More devices are available

- Create more "always available" resources
  - Turn on devices that are off
  - Minimize likelihood a device is unreachable
- Support users
  - Access data without pre-planning
  - Don't force or second-guess their choices

Combine new hardware with
best-practice storage system techniques

Outline

- Introduction and motivation
- **Design**
- Implementation and evaluation
- Conclusion

ZZFS: Design considerations

- Human factors
  - User studies to examine storage and access behavior
- Hardware
  - Low-power NIC to turn on devices (Agarwal ’09)
  - Additional temporary storage
- Storage system best practices
  - Versioned histories for consistency
  - I/O offloading when needed for performance

Human factors: Sources

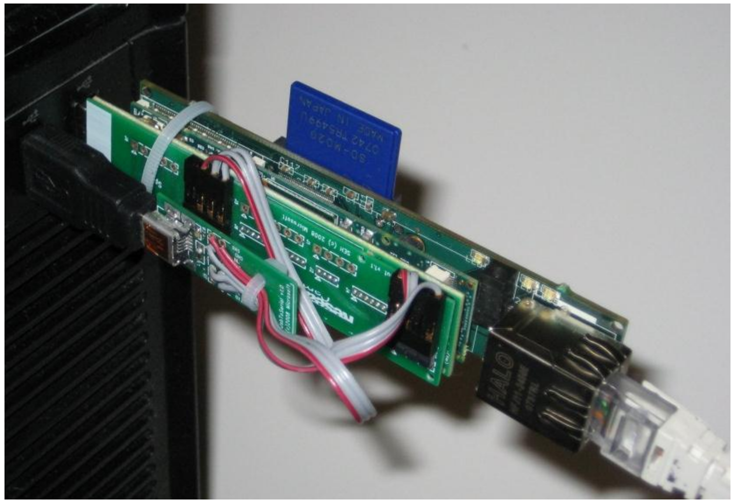
- Goal: Understand how and why users store, organize and access content across devices
  - Inform design decisions
- Sources:
  - Analysis of feedback from LiveMesh and Dropbox
  - Small-scale qualitative study
  - Large-scale qualitative study (Odom et al., CHI 2012)

Human factors: Findings

- Don’t require hoarding
  - People are busy
  - People don’t always know what they’ll need
- Placement is deliberate and reasoned
  - Desire to know and control where data is
  - Trust considerations for cloud storage (Odom 2012, Ion 2011)
- Don’t second-guess users’ decisions

Hardware: Somniloquy NIC

- Maintains access while PC is dormant
  - Wake up as needed ("always-on")
  - Transparent to applications
- 10-100x less power than idle PC
- On-board flash for temporary storage


Agarwal et al., NSDI 2009

Storage system

- Metadata service
  - How devices find the data
- I/O director
  - Sets policy for where/how data is found
- I/O offloading
- Example use cases

Storage system: Metadata service

- Flat, device-transparent namespace
- Can reside anywhere
  - Could replicate content or service
- Simple default for personal data
  - Single instance, cached on all devices
- Metadata size << data size
  - ≤ 0.1% (our families)
  - Consistent with Eyo (Strauss 2011)
  - Easily storable, cacheable

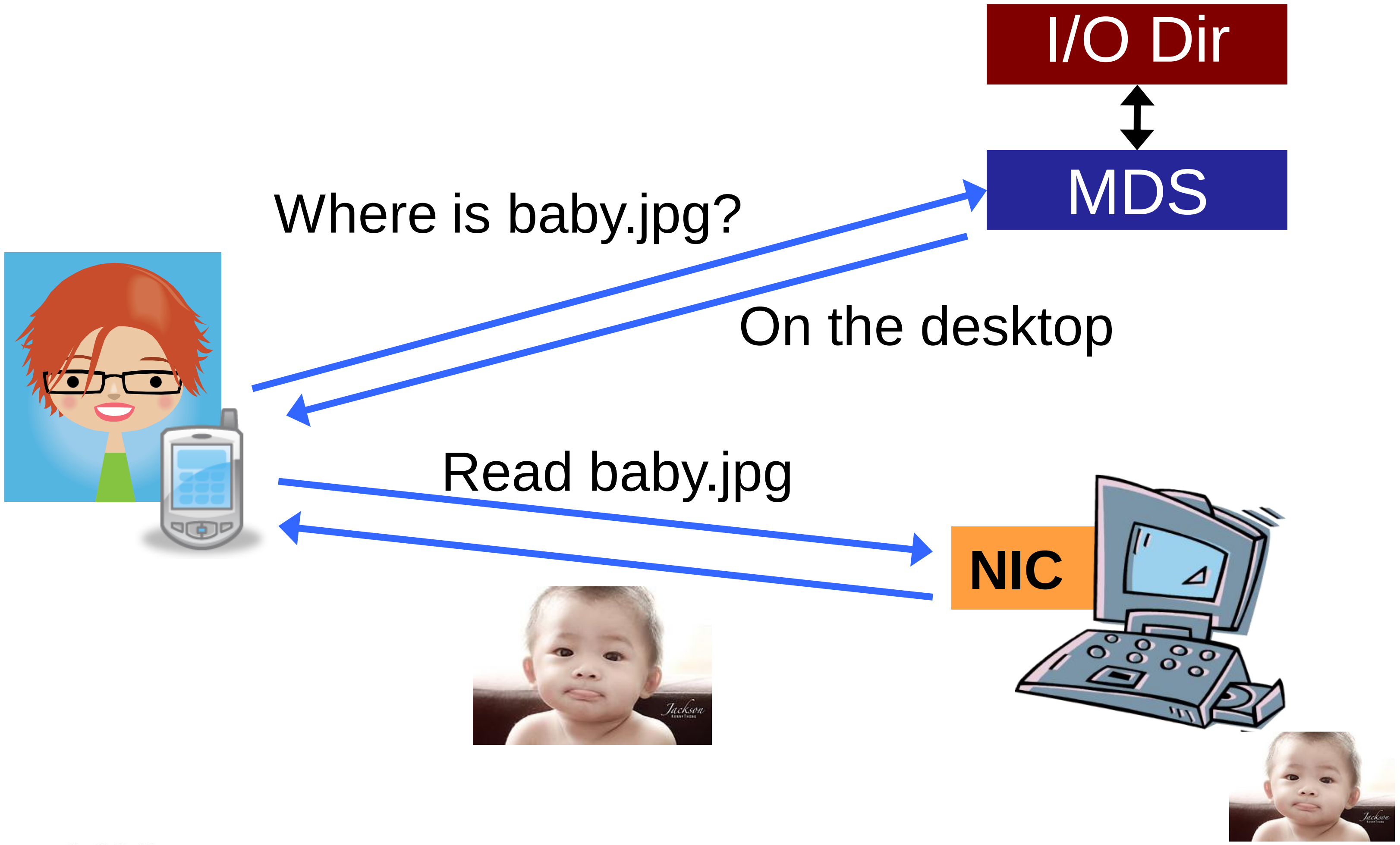
Storage system: I/O director

- Sits above metadata service
- Determines how data is stored, accessed
  - Optimize for energy, cost, latency, etc.
- New options for placement, access
  - Wake device when dormant
  - Offload writes when unavailable
    - To on-board flash, cloud, other devices

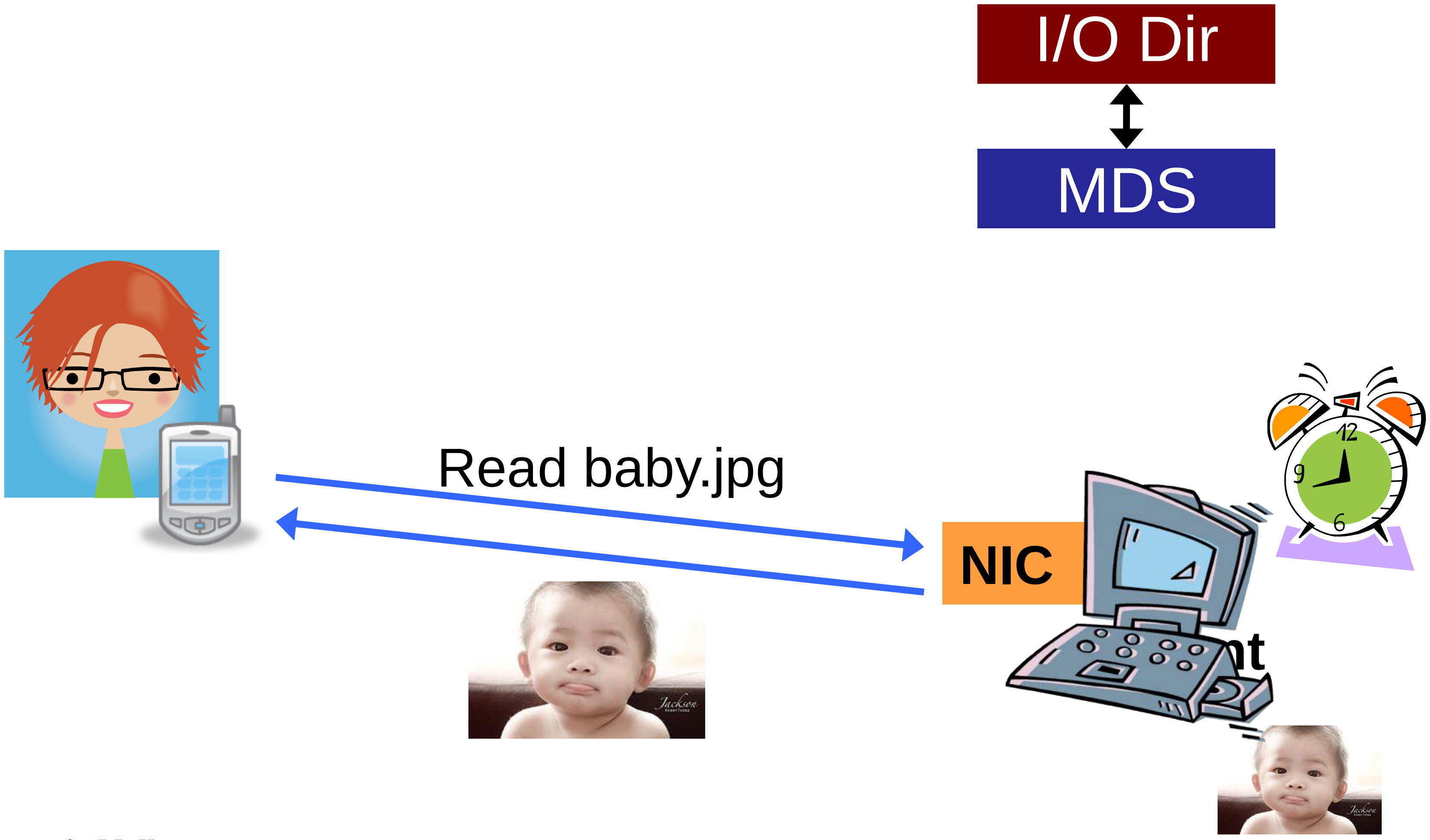
Storage system: I/O offloading

- Builds on past work (Everest, Sierra)
- Offload updates to spare resources as needed
- All offloaded data eventually reclaimed
- Allows online data migration

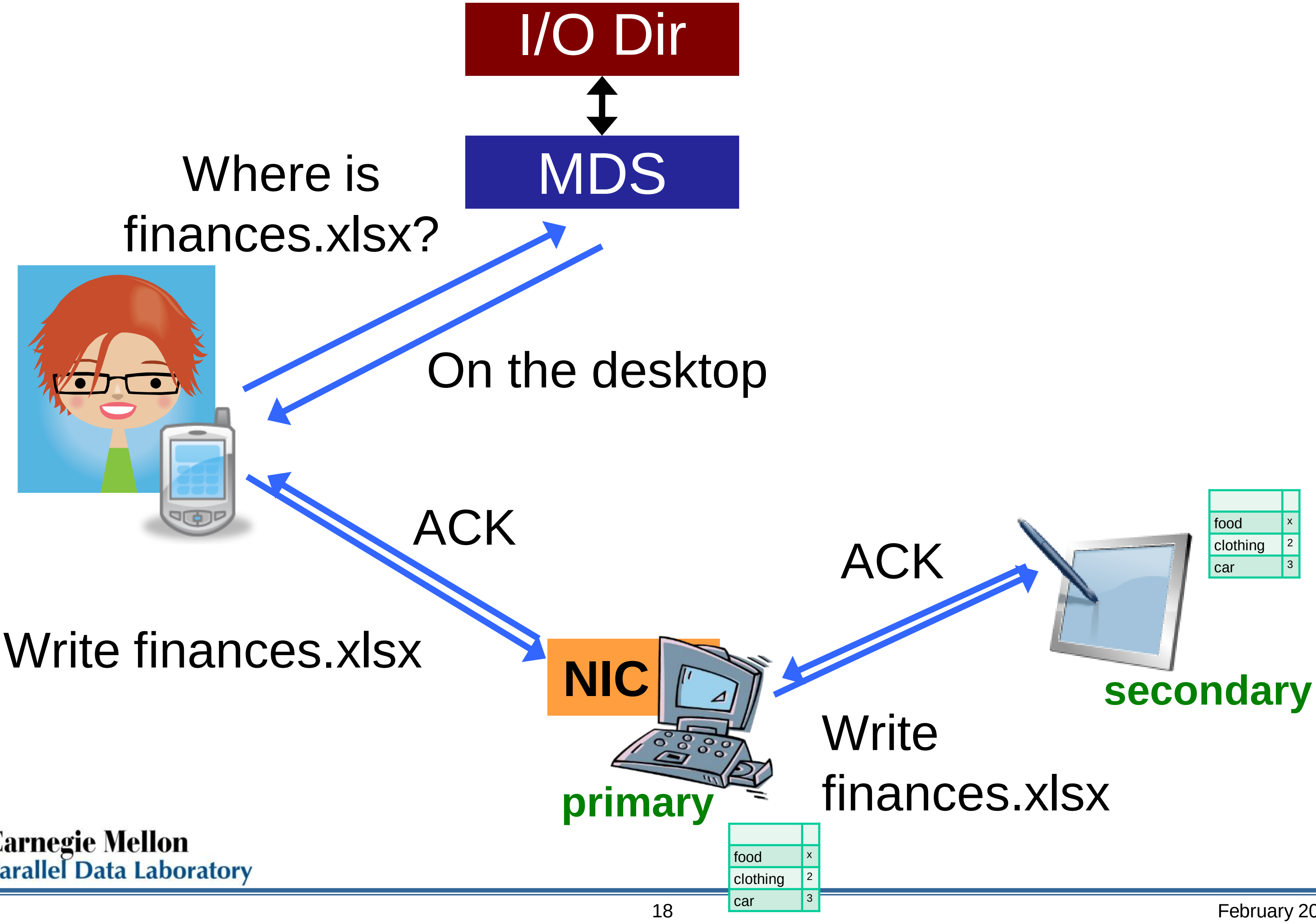
Standard read



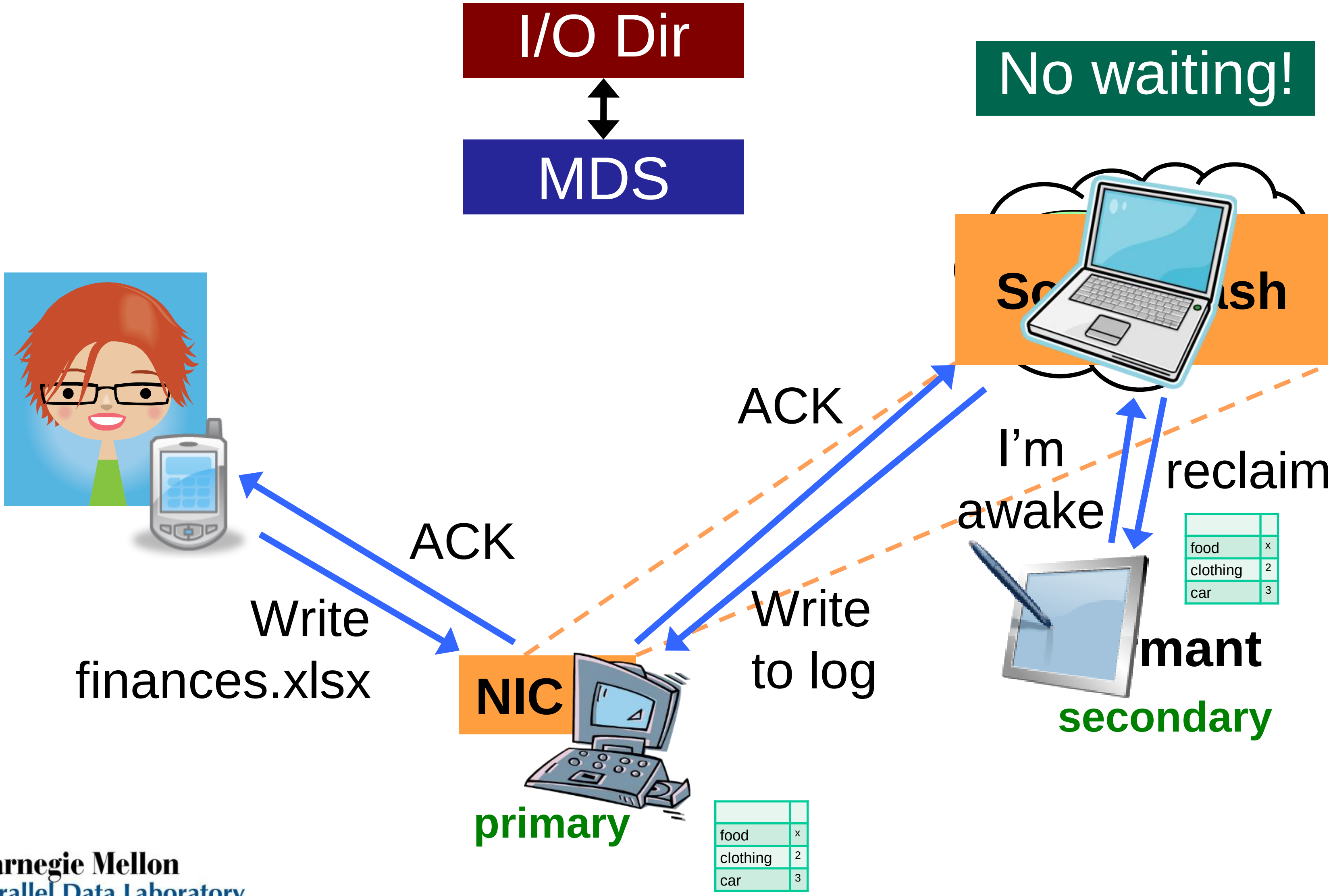
Read while dormant: Wake up



Standard write



Write with offload/reclaim



Carnegie Mellon
Parallel Data Laboratory

Other placement options

- Write with no network connection:
  - Offload all remote writes to local log
  - Upon connection, implicated devices reclaim
  - Standard conflict resolution (e.g. Bayou)
- Move primary before device sleeps
- More in the paper

Other design considerations

- Broadband at home, weak 3G while mobile
- Somniloquy is a prototype
- Application to mobile devices, tablets

Outline

- Introduction and motivation
- Design
- **Implementation and evaluation**
- Conclusion

Implementation

- Simple application of always-on design
- User level, in C, above NTFS or FAT
- Run legacy applications via WebDav detours
- Per-object replication
- Default placement: 1R, leave where created

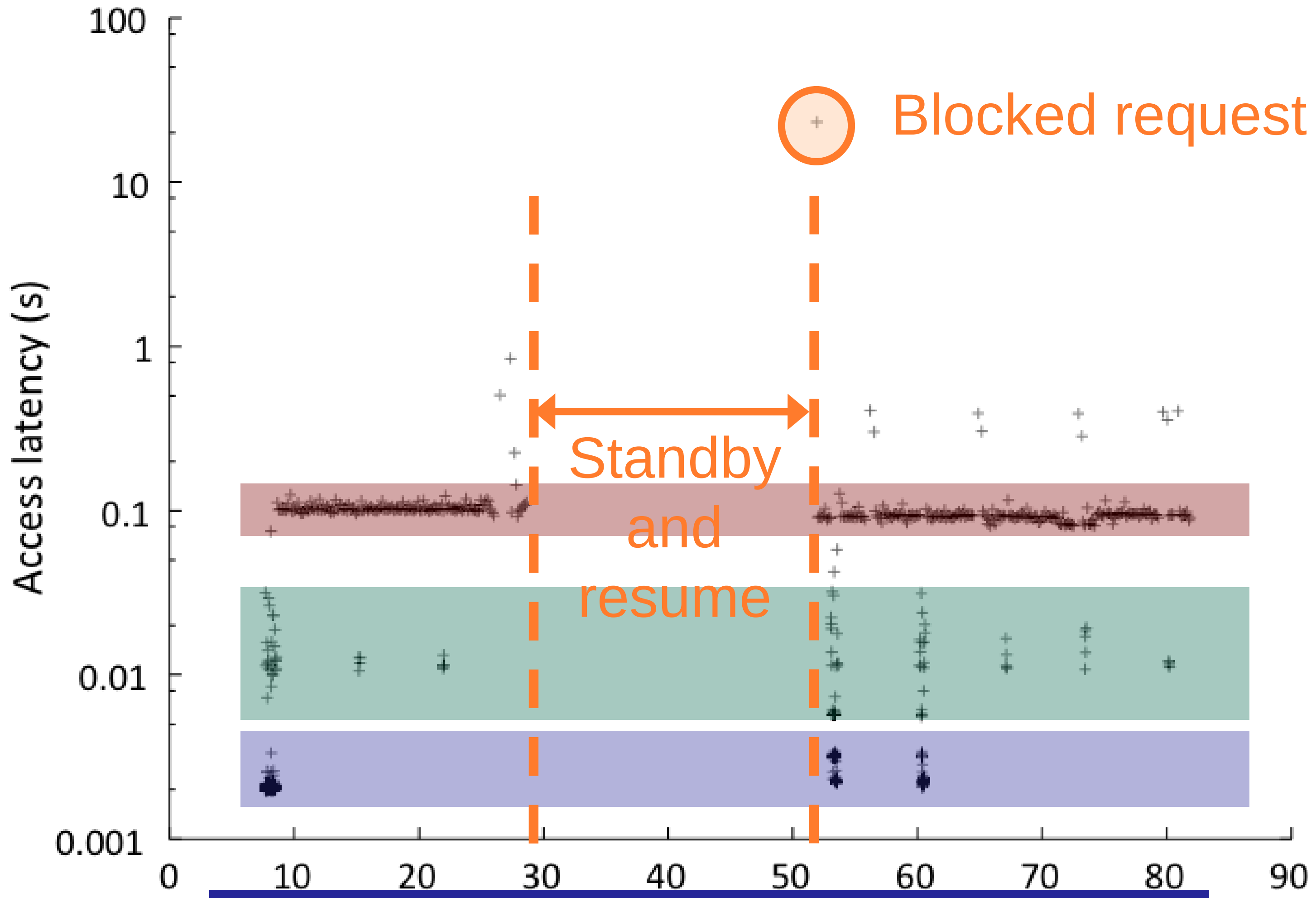
Evaluation overview

- Throughput: Low overhead
- **Read latency: Standby penalty**
- **Write latency: Standby penalty**
- Access latency and placement policy
- Moving files: Limited performance penalty
- Sensitivity to parameters
  - Fraction of local vs. remote accesses
  - Fraction of reads vs. writes

Read latency: Music example

- User listening to music
  - Songs split local and remote, shuffle mode
  - No replication
  - Read entire song in chunks; then small db write
- Goal: Evaluate worst case
  - Read request arrives just as shutting down

Read latency: Music example



Worst case: Performance is OK

Remote small writes

Remote reads

Carnegie Mellon
Parallel Data Laboratory

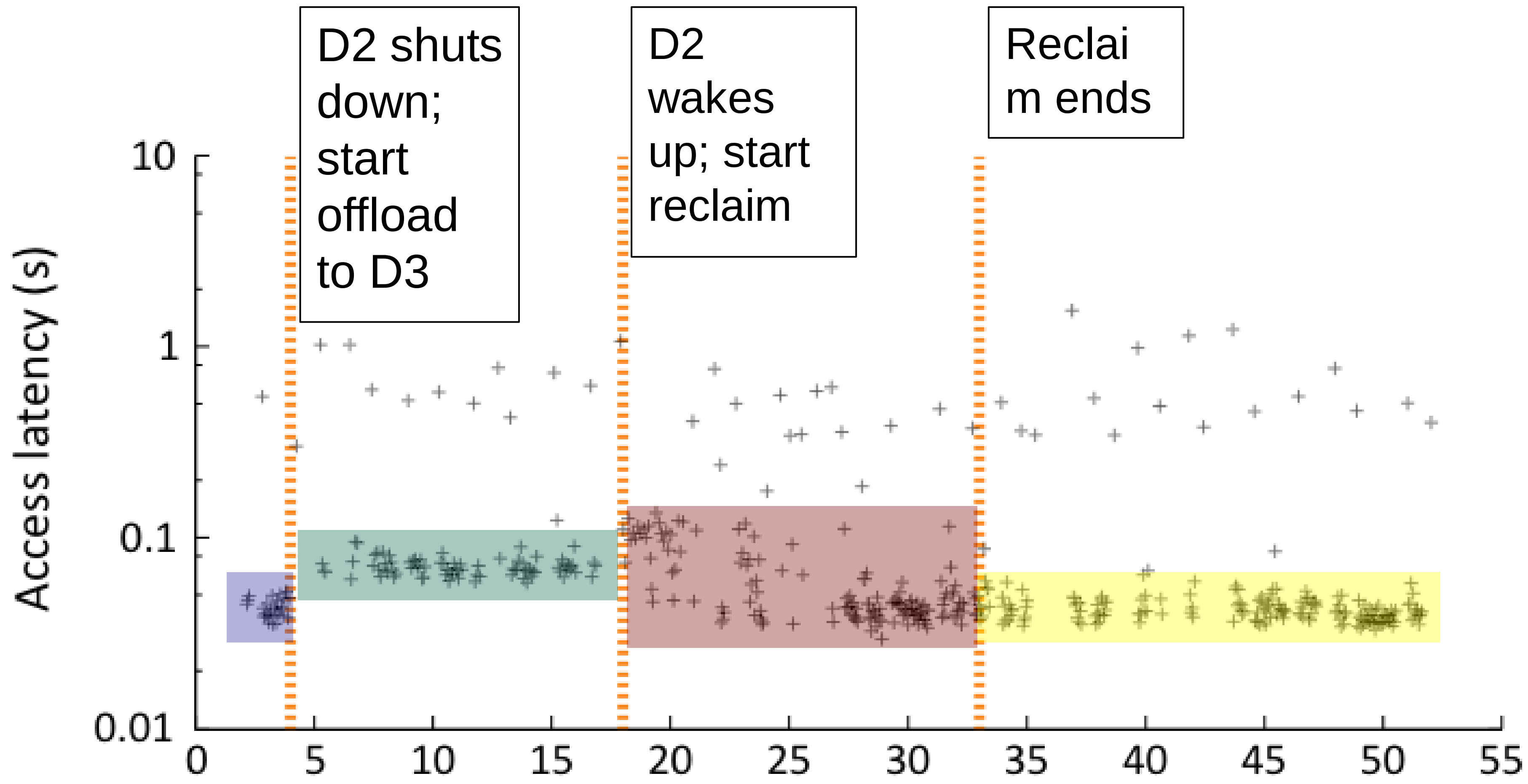
Latency: Standby and resume

Device	Standby (s)	Resume (s)
Lenovo x61 (Win7)	3.8	2.6
Dell T3500 (Win7)	8.7	7.2
HP Pavilion (XP)	4.9	10.3
Macbook Pro (OSX 10.6.8)	1.0	2.0
Ubuntu 11.10	11.0	4.5

Write latency: Document example

- Writes to an office document
  - 2 replicas: D1(local) and D2 (remote)
- Worst case: remote device goes into standby
  - Offload to D3 while it resumes
  - When it’s awake, reclaim
- **Unlike for read, offload masks switch-on**

Write latency: Document example



Offloading masks switch-on cost

Conclusion

- ZZFS makes spontaneous access to distributed content work better
  - Low-power, always-on comm channel
  - Helps execute placement policies
  - Helps compensate for uncertainties in device availability, user behavior
  - Accounts for human factors