Using volunteered resources for data-intensive computing and storage

David Anderson

Space Sciences Lab
UC Berkeley

10 April 2012

The consumer digital infrastructure

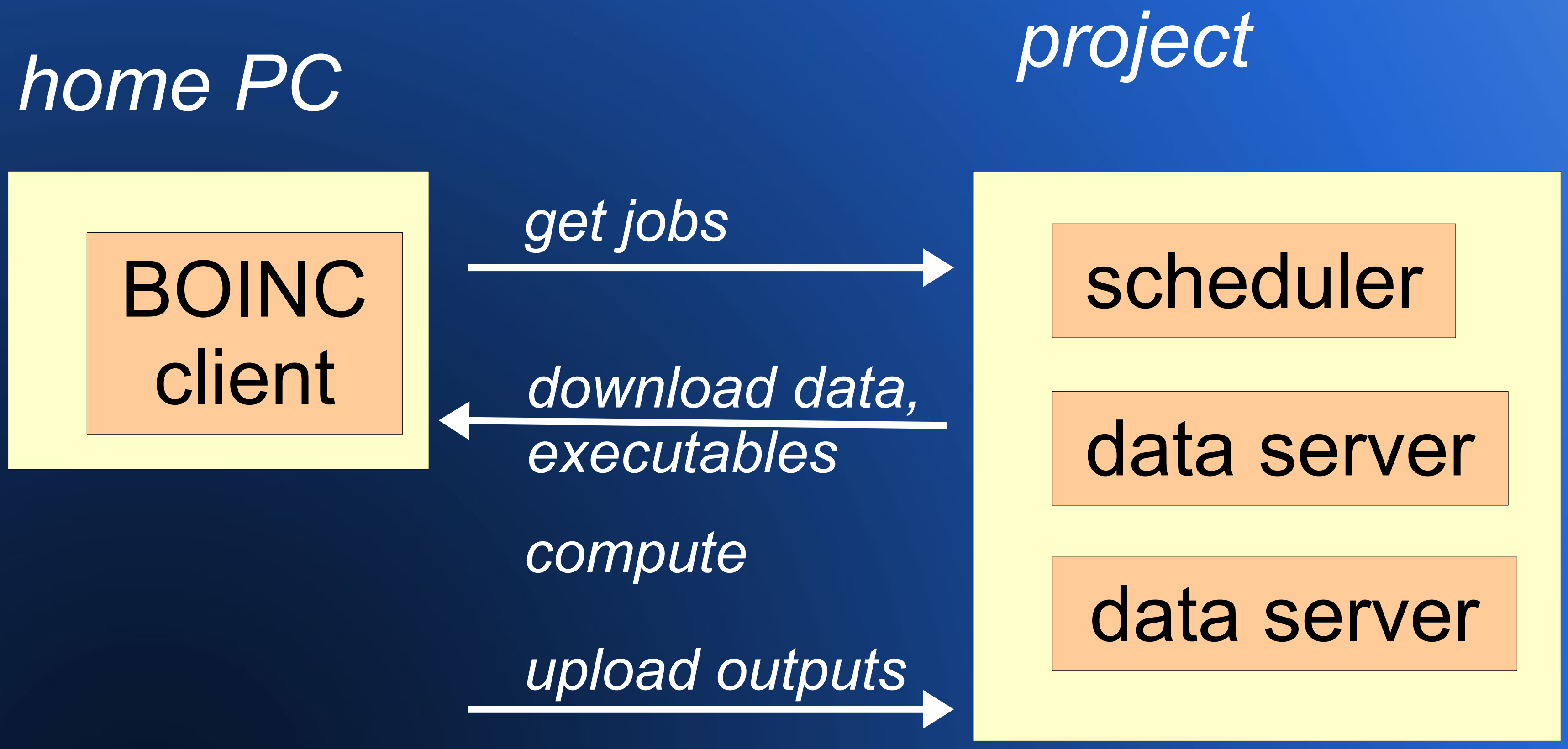
Consumer

- 1.5 billion PCs
  - GPU-equipped
  - paid for by owner
- 5 billion mobile devices
- Commodity Internet

Organizational

- 2 million cluster/cloud nodes (no GPUs)
- supercomputers
- Research networks

Volunteer computing



- Clients can be attached to multiple projects

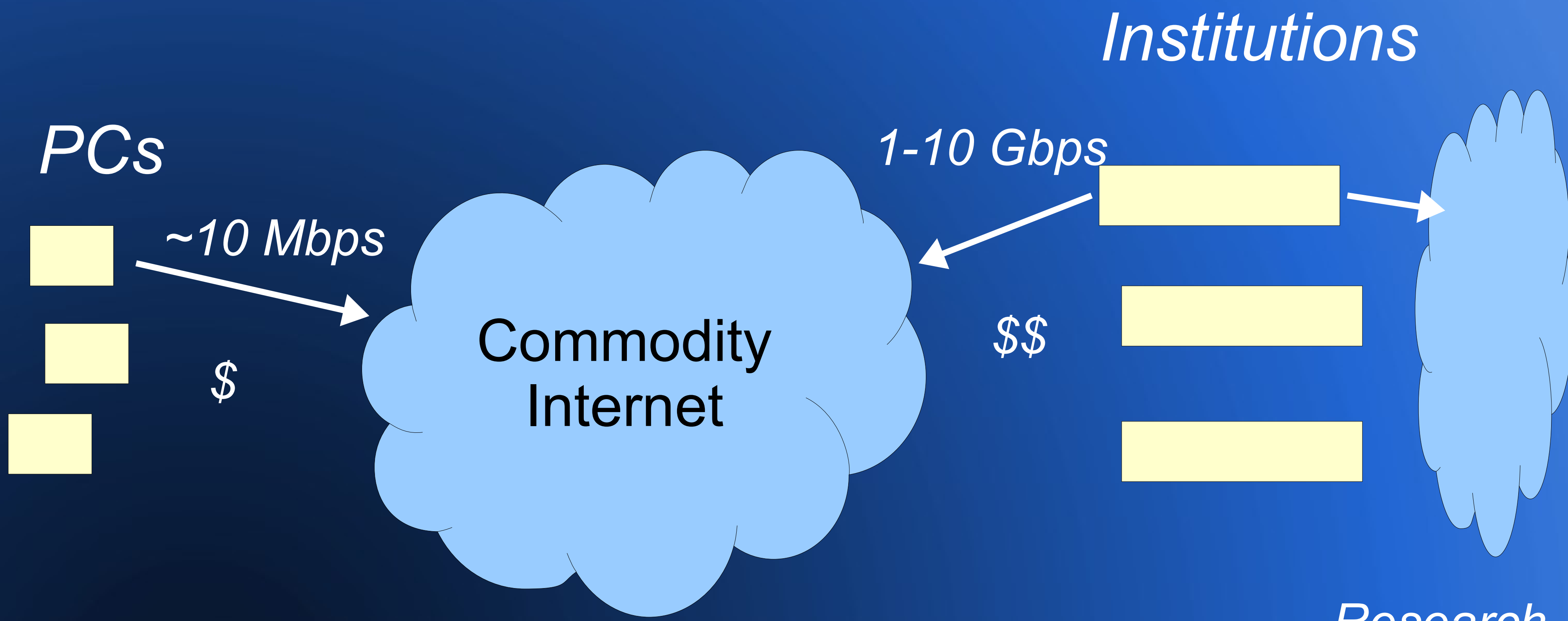
Volunteer computing status

- 700,000 active PCs
  - 50% with usable GPUs
- 12 PetaFLOPS actual throughput
- Projects
  - CAS@home
  - IBM World Community Grid
  - Einstein@home
  - ... 50 others

Data-intensive computing

- Examples
  - LHC experiments
  - Square Kilometer Array
  - Genetic analysis
  - Storage of simulation results
- Performance issues
  - network
  - storage space on clients

Networking landscape



- Most PCs are behind NATs/firewalls
  - only use outgoing HTTP

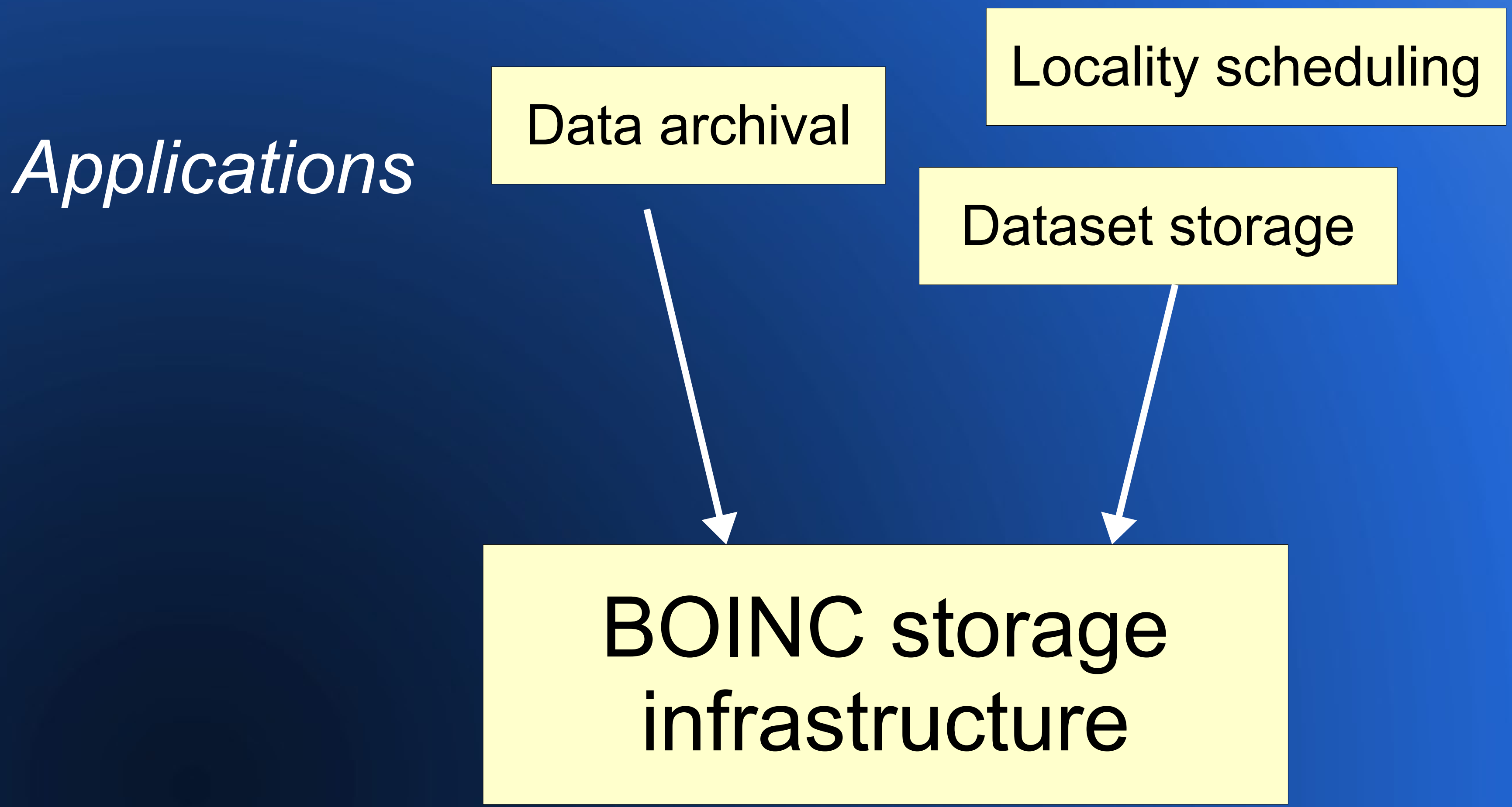
Disk space on clients

- Current
  - average 50 GB available per client
  - 35 PetaBytes total
- Trends
  - disk sizes increasing exponentially, faster than processors
  - 1 TB * 1M clients = 1 Exabyte

Properties of clients

- Availability
  - hosts may be turned off
  - hosts may be unavailable by user preference
    - time of day
    - PC is busy or not busy
- Churn
  - The active life of hosts follows an exponential distribution with mean ~100 days
- Heterogeneity
  - wide range of hardware/software properties

BOINC storage architecture



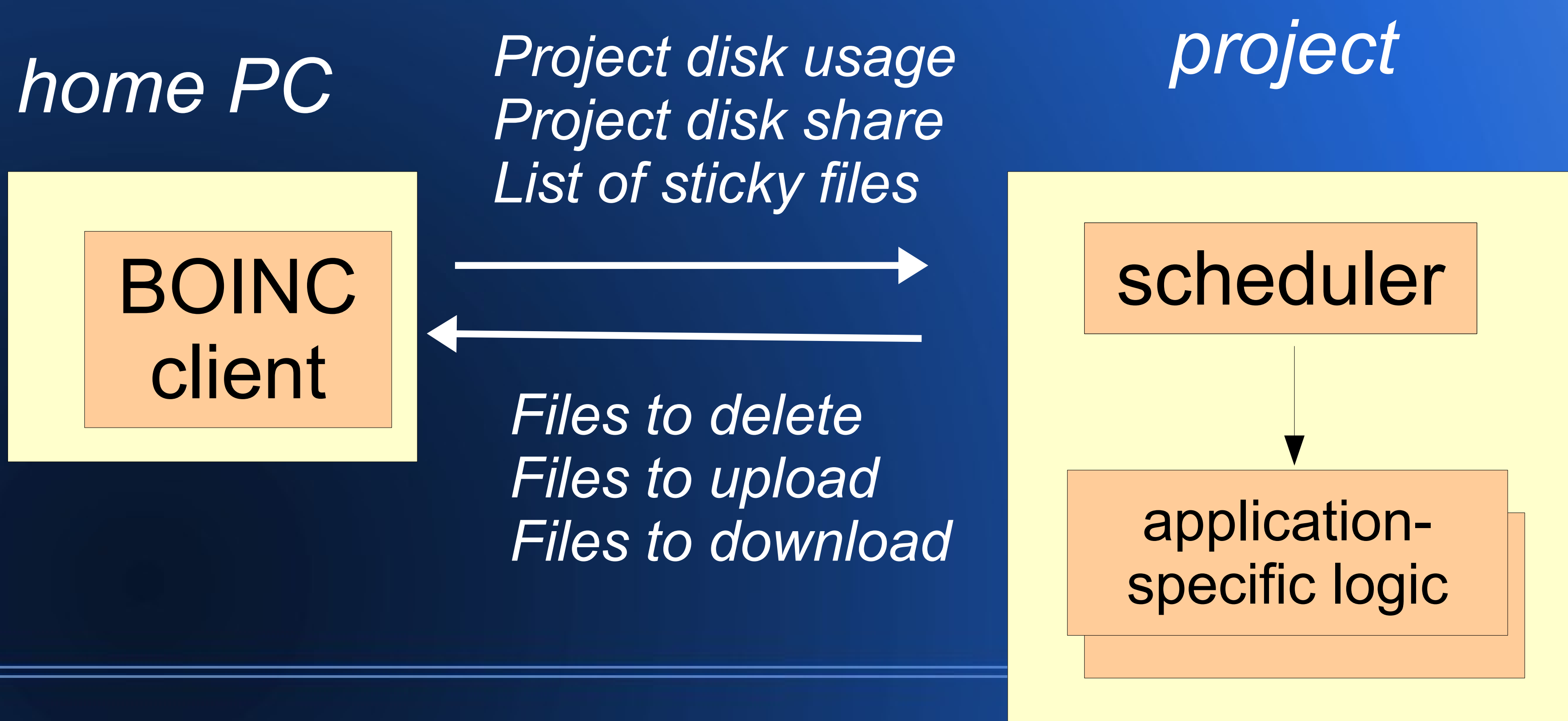
BOINC storage infrastructure: managing client space

- Volunteer preference: keep at least X% free

- This determines BOINC’s total allocation
- Allocation among projects is based on volunteer-specified “resource share”

BOINC storage infrastructure: file management

- “Sticky file” mechanism



Storage applications

- Temporary storage of simulation results
- Dataset storage
- Locality scheduling
- Data archival

Temporary storage of simulation results

- Many simulations (HEP, molecular, climate, cosmic) produce small “final state” file, large “trajectory” file.
- Depending on contents of final state file, scientist may want to examine the trajectory file
- Implementation
    - make trajectory files sticky, non-uploaded
    - interface for uploading trajectory files
    - Interface for retiring trajectory files

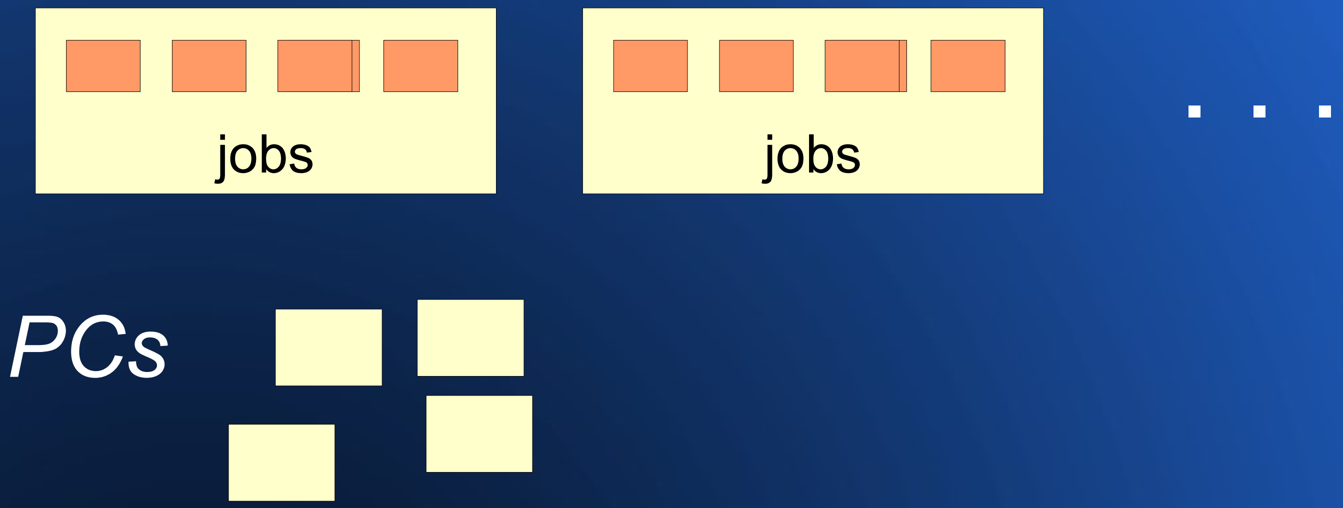
Dataset storage

- Goal:
  - submit queries against a dataset cached on clients (main copy is on server)
  - Minimize turnaround time for queries
- Scheduling policies
  - whether to use a given host
  - how much data to store on a given host
    - should be proportional to its processing speed
  - update these decisions as hosts come and go

Locality scheduling

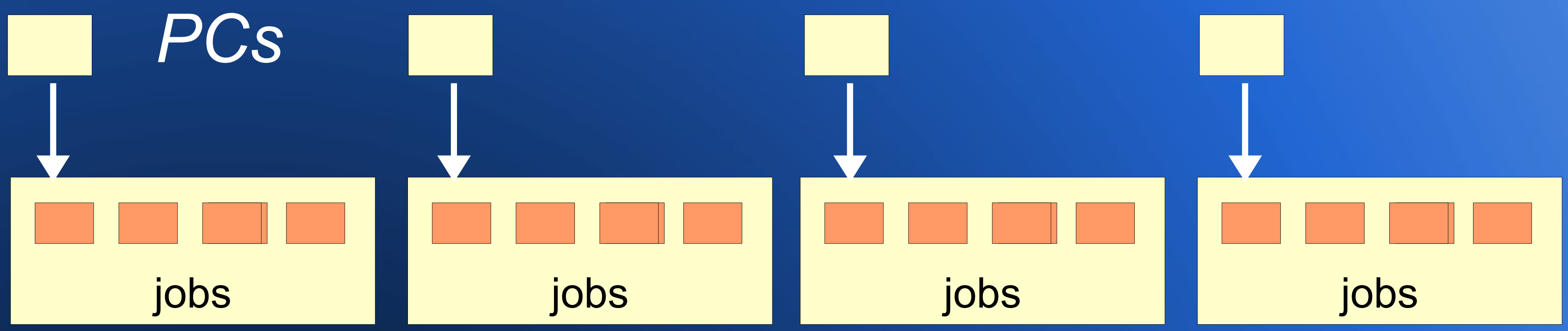
- Have a large dataset
- Each file in the dataset is input for a large number of jobs
- Goal: process the dataset using the least network traffic
- Example: Einstein@home analysis of LIGO gravity-wave detector data

Locality scheduling



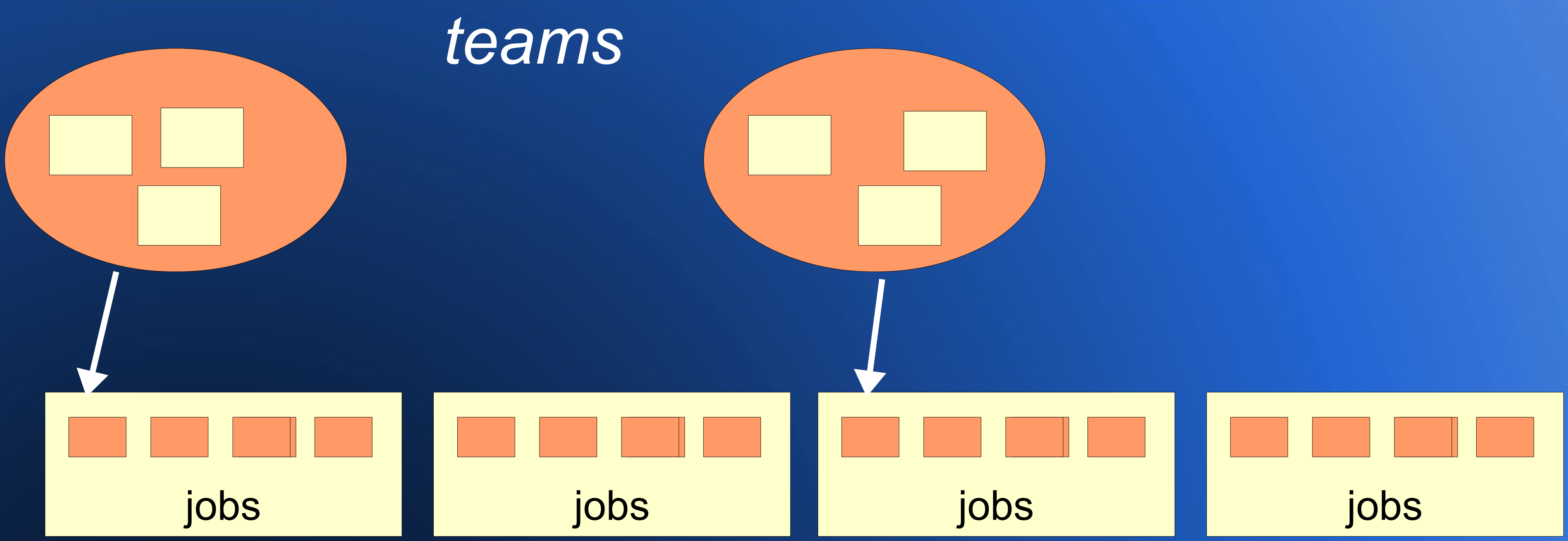
- Processing jobs sequentially is pessimal
  - every file gets sent to every client

Locality scheduling: ideal



- Each file is downloaded to 1 host
- Problems
  - Typically need job replication
  - Widely variable host throughput

Locality scheduling: actual



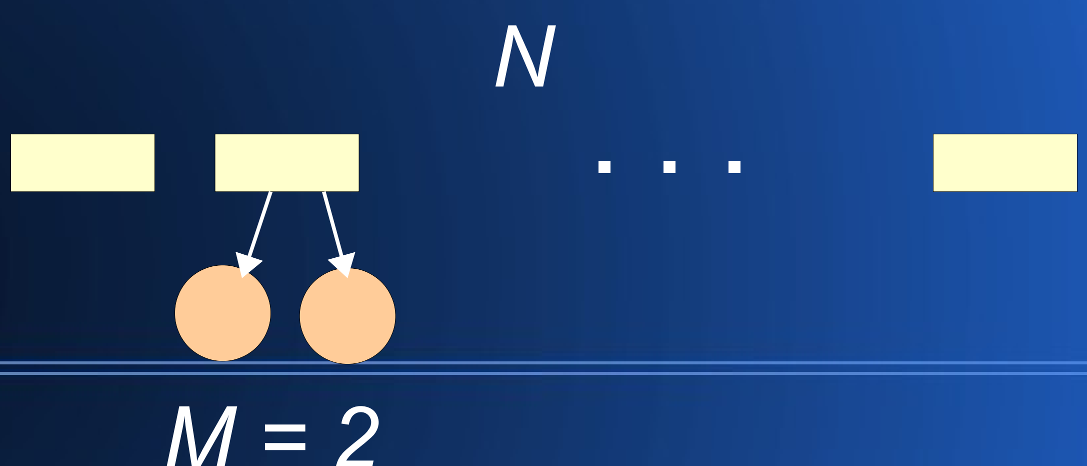
- New hosts are assigned to slowest time
- Teams are merged when they collide
- Each file is downloaded to ~10 hosts

Data archival

- Files originate on server
- Chunks of files are stored on clients
- Files can be reconstructed on server (with high latency)
- Goals:
  - arbitrarily high reliability (99.999)
  - support large files

How to achieve reliability?

- Replication
  - Divide file into N chunks
  - Store each chunk on M clients
  - If a client fails
    - upload another replica to server
    - download to a new client



Problems with replication

- Hard to achieve high reliability

  C = probability of losing a particular chunk

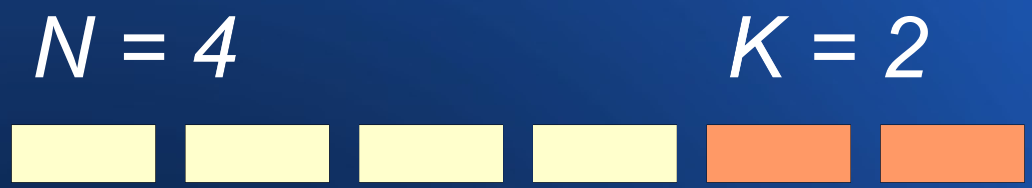
  F = probability of losing some chunk

  $F = 1 - (1-C)^N$

  $0.36 = 1 – (1-0.0001)^{1000}$
- High space overhead
  - use Mx space to store an x-byte file

Reed-Solomon Coding

- A way of dividing a file into N+K chunks

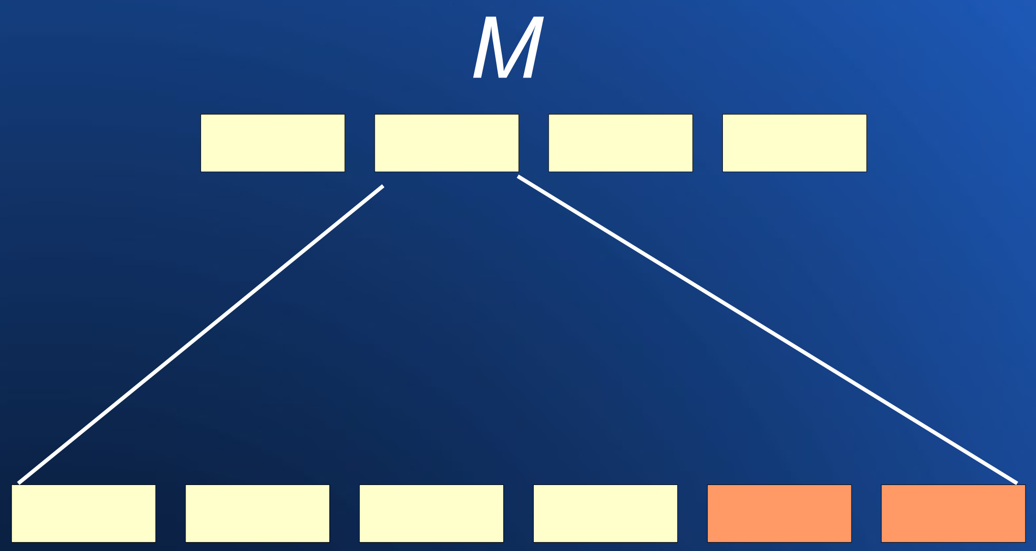


- The original file can be reconstructed from any N of these chunks.
- Example: N=40, K=20
  - can tolerate simultaneous failure of 20 clients
  - space overhead is only 50%

The problem with coding

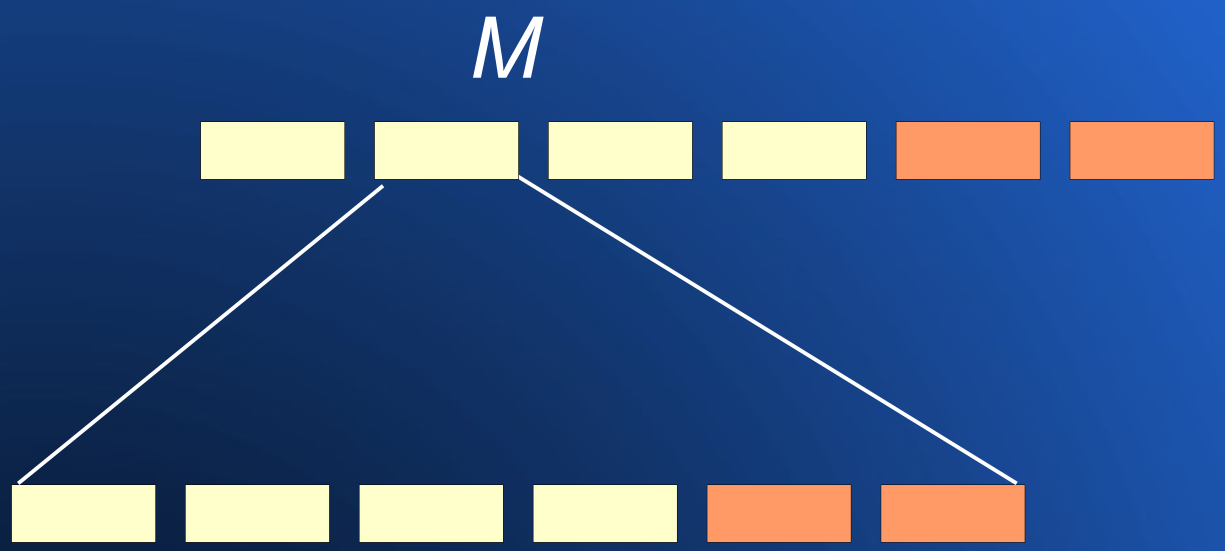
- When any chunk fails, need to upload all other chunks to server
- High network load at server
- High transient disk usage at server

Reducing coding overhead



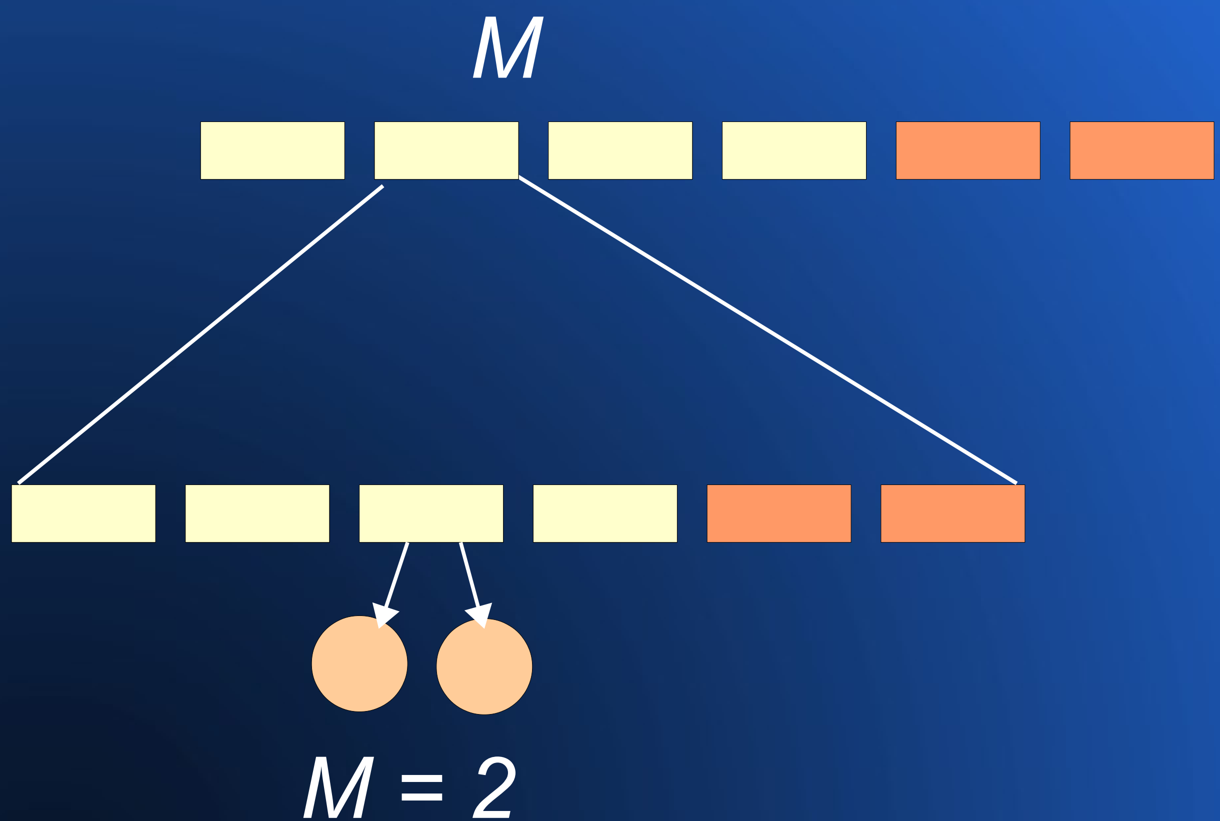
- Only need to upload 1/M of file on failure

Two-level coding



- Can tolerate K^2 client failures
- Space overhead: 125%

Two-level coding + replication



- Most recoveries involve only 1 chunk
- Space overhead: 250%

Volunteer storage simulator

- Predicts the performance of coding/replication policies
- Inputs:
  - description of host population
  - policy, file size
- Outputs:
  - disk usage at server
  - upload/download traffic at server
  - fault tolerance level

Implementation status

- Storage infrastructure: done (in 7.0 client)
- Storage applications:
  - Data archival: 1-2 months away
  - Locality scheduling:
    - used by Einstein@home, but need to reimplement
  - Others: in design stage

Conclusion

- Volunteer computing can be data-intensive
  - With 200K clients, could handle the work of all LHC Tier 1 and 2 sites
  - In 2020, can potentially provide Square Kilometer Array (1 Exabyte/day) with 100X more storage than on-site resources
- Using coding and replication, we can efficiently transform a large # of unreliable storage nodes into a highly reliable storage service

Future work

- How to handle multiple competing storage applications within a project?
- How to grant credit for storage?
  - how to make it cheat-proof?
- How to integrate peer-to-peer file distribution mechanisms
  - Bittorrent, Attic