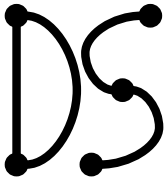


Research Group
Distributed Systems

Distributed Systems

TOR & GFS

Olaf Landsiedel

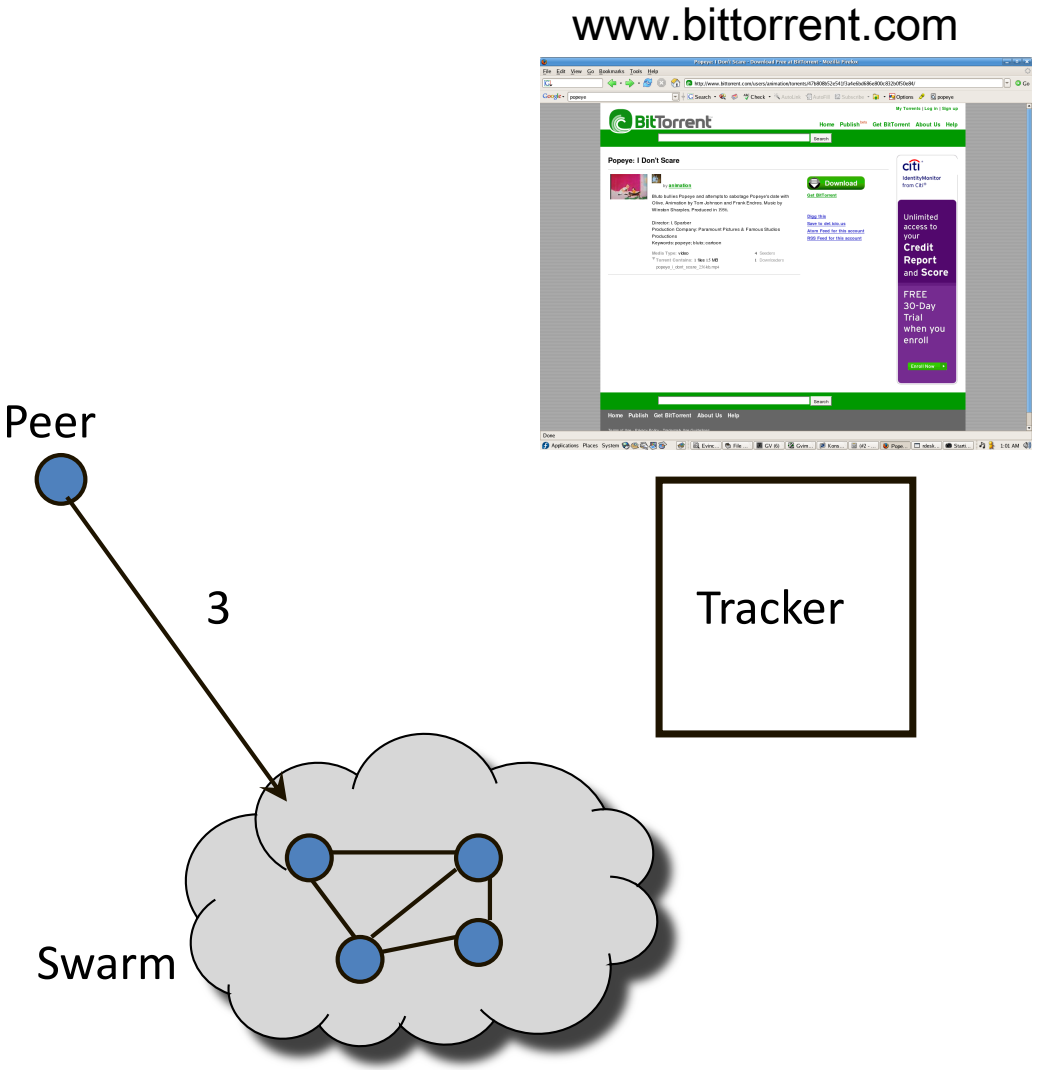
Last Time

- Blockchains III
  - Smart Contracts
  - Ethereum
  - Public Blockchains
  - Hyperleder

Today

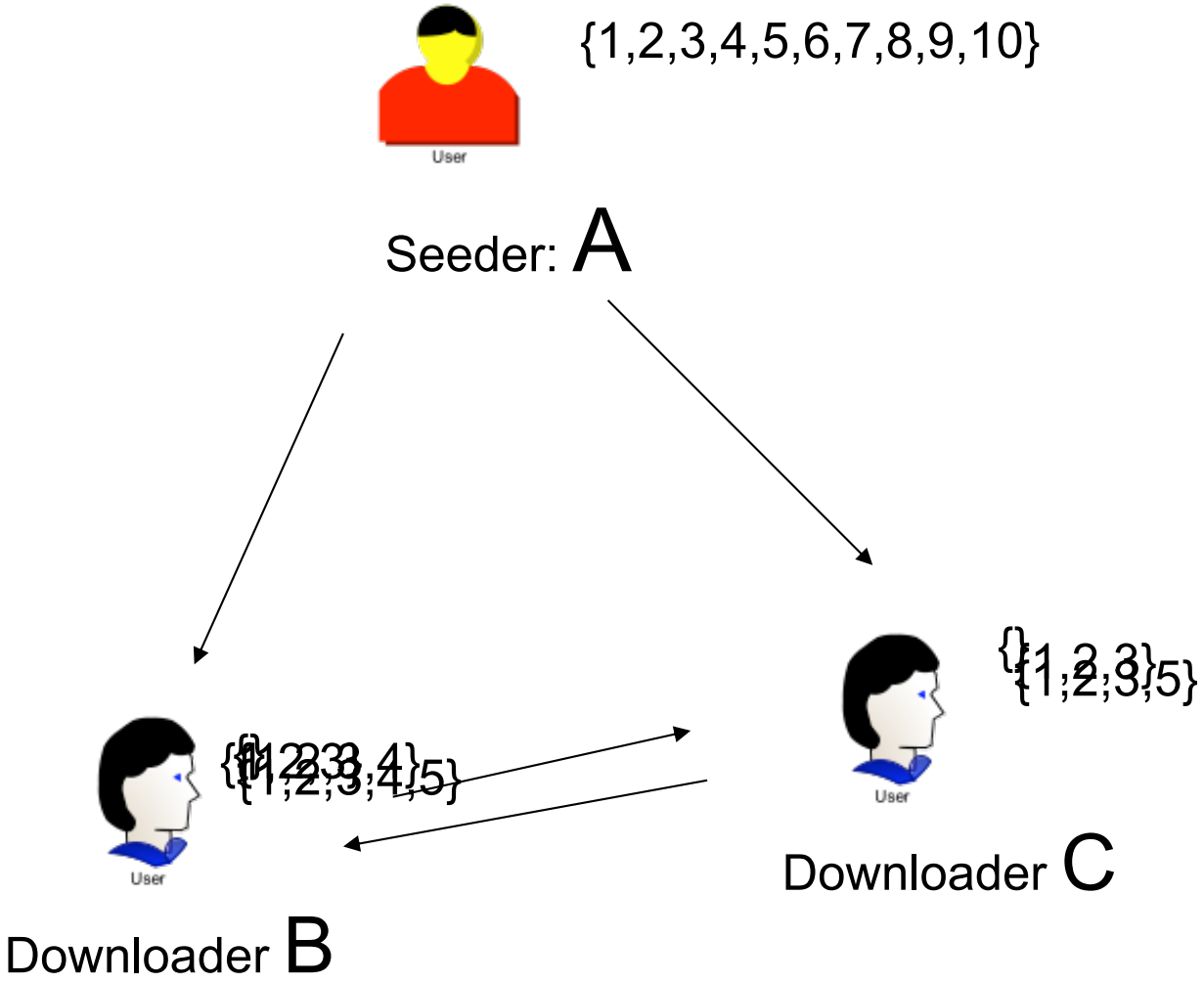
- Compelte the Dark Side
  - Finish Bittorent
  - TOR
- White Side
  - Google File System
  - And more…

How a node enters a swarm for file “popeye.mp4”



- File popeye.mp4.torrent hosted at a (well-known) webserver
- The .torrent has address of tracker for file
- The tracker, which runs on a webserver as well, keeps track of all peers downloading file -> swarm

Simple example



Problem

- Single point of failure

Pros and cons of BitTorrent

- Dependence on centralized tracker: pro/con?
  - ☹ Single point of failure: New nodes cannot enter swarm if tracker goes down
  - Lack of a search feature
    - ☺ Prevents pollution attacks
    - ☹ Users need to resort to out-of-band search: well known torrent-hosting sites / plain old web-search

“Trackerless” BitTorrent

- To be more precise, “BitTorrent without a centralized-tracker”
- Uses a **Distributed Hash Table** (Kademlia DHT)
  - Key, value pair: “tracker file name”, “tracker file”
  - Use simple put/get to upload/retrieve file
  - DHT is resilient to node failure: uses replication etc.
- Tracker DHTs run by normal end-hosts
  - Not a web-server anymore
    - Remove single point of failure
  - Node responsible for its id in the DHT
    - As discussed for CAN, Chord etc.

BitTorrent: Ethical Challenges

- Ethical challenges?
  - Use to distributed legal and illegal content
  - Hard to remove illegal content
    - No central authority / point of control
    - Pro's & cons of this design
      - Hard to remove illegal content
        - » Copyrighted movies etc.
      - Hard to remove content that some governments etc. do not want to be distributed
        - » Documentations of corruption etc.

BitTorrent

- BitTorrent Summary
  - Basic
  - Tit-for-tat
  - Trackerless

- http://olafland.polldaddy.com/s/bittorrent
  - In BitTorrent you can only share legal data
  - In BitTorrent you can search for the content you are interested in
  - BitTorrent is scalable
  - BitTorrent: Each swarm has a single point of failure
  - “Trackerless” BitTorrent: Each swarm has a single point of failure
  - Free riding in BitTorrent

BitTorrent

- BitTorrent Summary
  - Basic
  - Tit-for-tat
  - Trackerless

- http://olafland.polldaddy.com/s/bittorrent
  - In BitTorrent you can only share legal data
    - no
  - In BitTorrent you can search for the content you are interested in
    - no
  - BitTorrent is scalable
    - yes
  - BitTorrent: Each swarm has a single point of failure
    - Yes, the tracker
  - “Trackerless” BitTorrent: Each swarm has a single point of failure
    - No, as now the tracker is in the DHT which includes redundancy
  - Free riding in BitTorrent
    - Possible to a (very) small amount (hope for optimistic unchoking)

Today

- TOR
- Google File System

TOR

TOR: The Onion Router

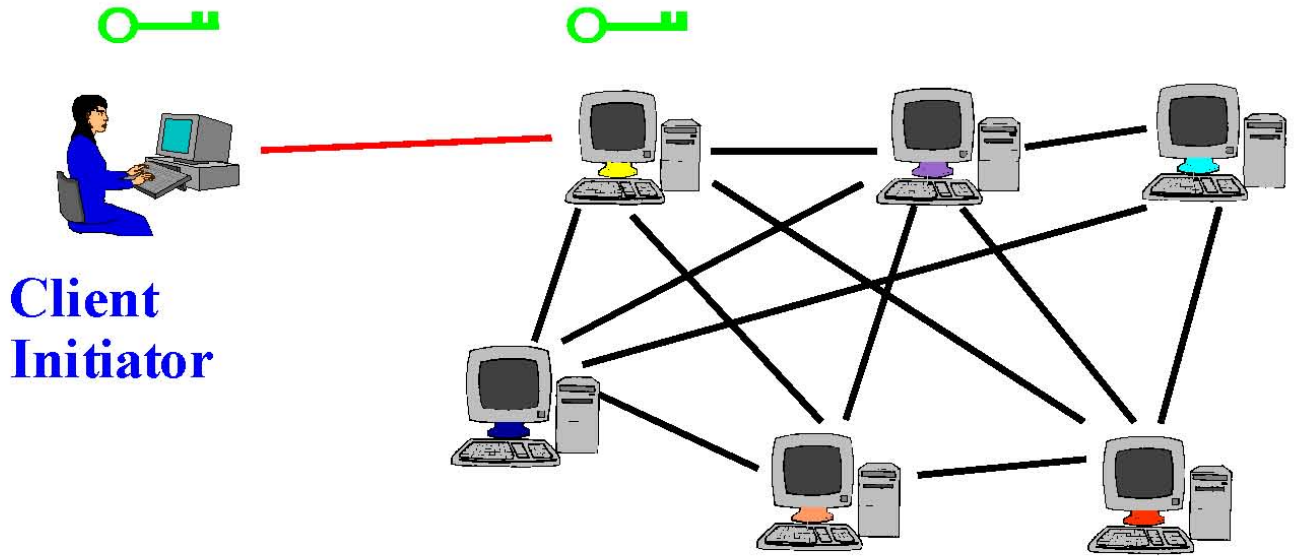
- Have you heard about TOR before?
- What does it do?

TOR

- Anonymous Internet
  - Based on onion routing
  - http://torproject.org
  - Specifically designed for low-latency anonymous Internet communications: web browsing etc.
- Running since October 2003
  - Thousands of relay nodes, 100K-500K? of users
- Easy-to-use client proxy, integrated Web browser

Tor Circuit Setup (1)

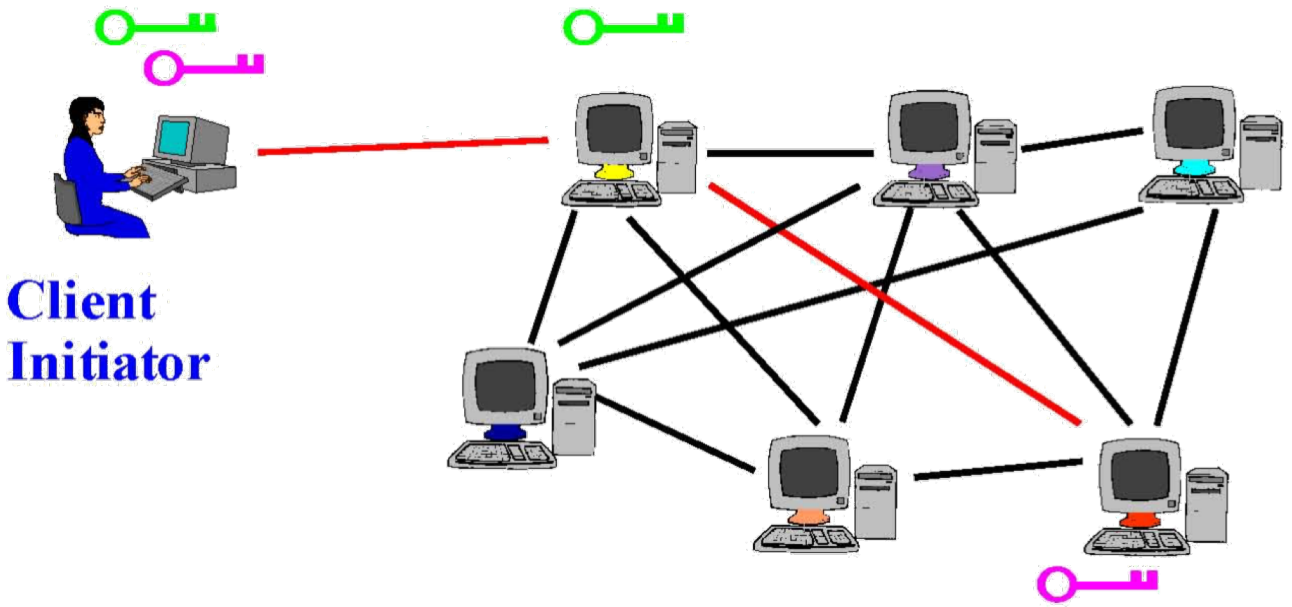
- Client proxy establishes a symmetric session key and circuit with relay node #1



End-point of a encrypted tunnel
-> take security courses for details ☺

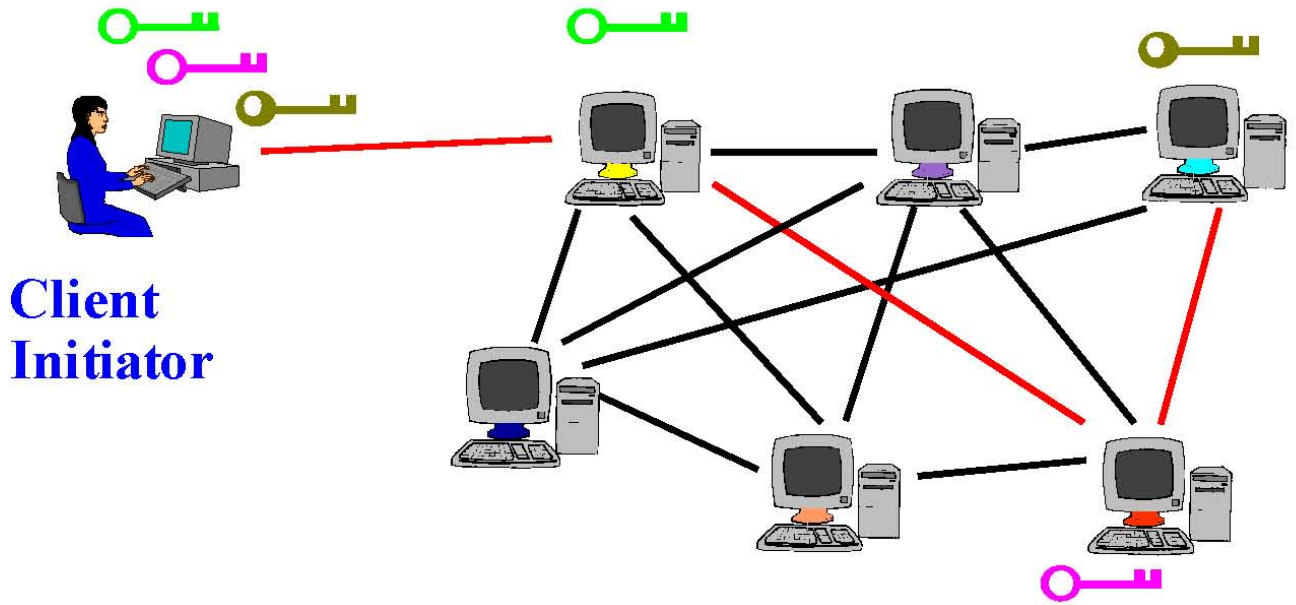
Tor Circuit Setup (2)

- Client proxy extends the circuit by establishing a symmetric session key with relay node #2
  - Tunnel through relay node #1



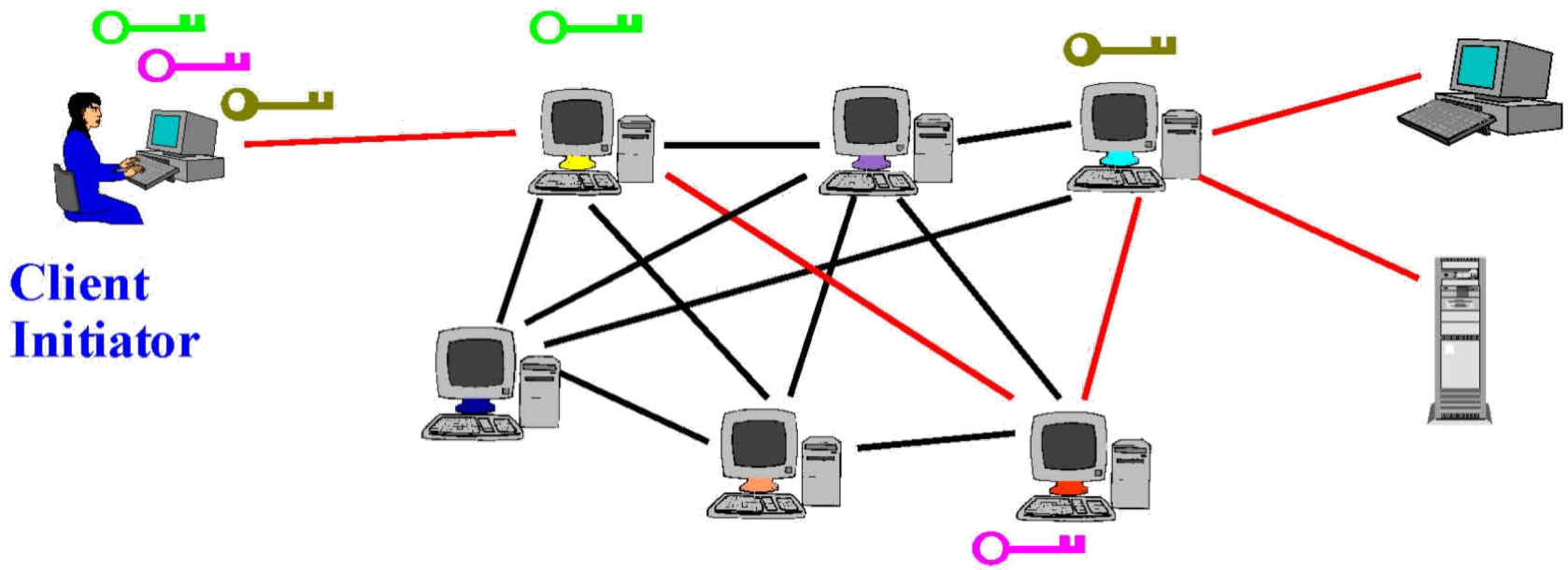
Tor Circuit Setup (3)

- Client proxy extends the circuit by establishing a symmetric session key with relay node #3
  - Tunnel through relay nodes #1 and #2



Using a Tor Circuit

- Client applications connect and communicate over the established Tor circuit
  - Circuit: anonymous tunnel between client and Internet
  - Layered encryption
  - Each node only knows about each predecessor and successor



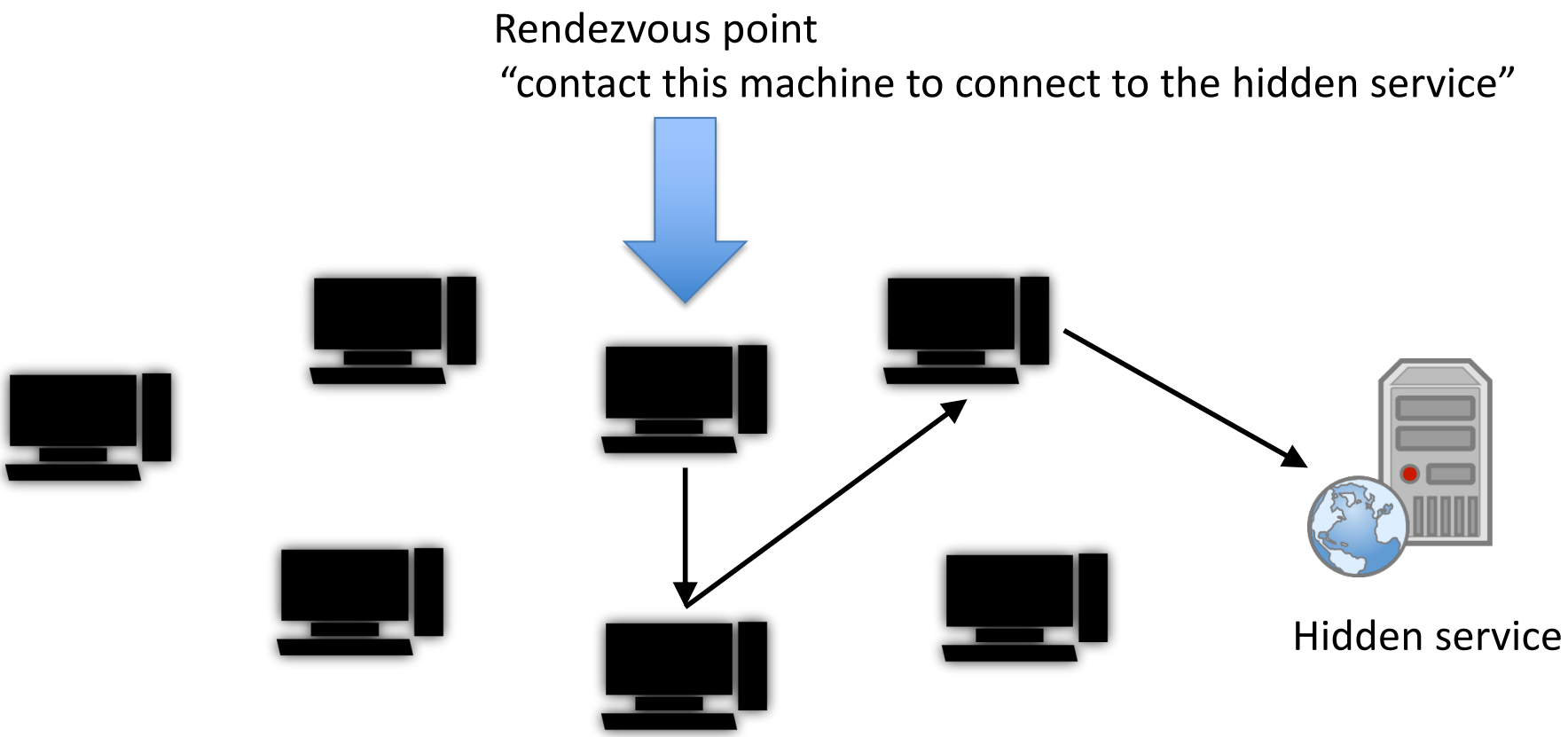
Using Tor

- Many applications can share one circuit
  - Multiple TCP streams over one anonymous connection
- Directory servers
  - Maintain lists of active relay nodes, their locations, current public keys, etc.
  - Directory servers' keys ship with Tor code
- Who provides the onion routers?
  - Mainly normal users

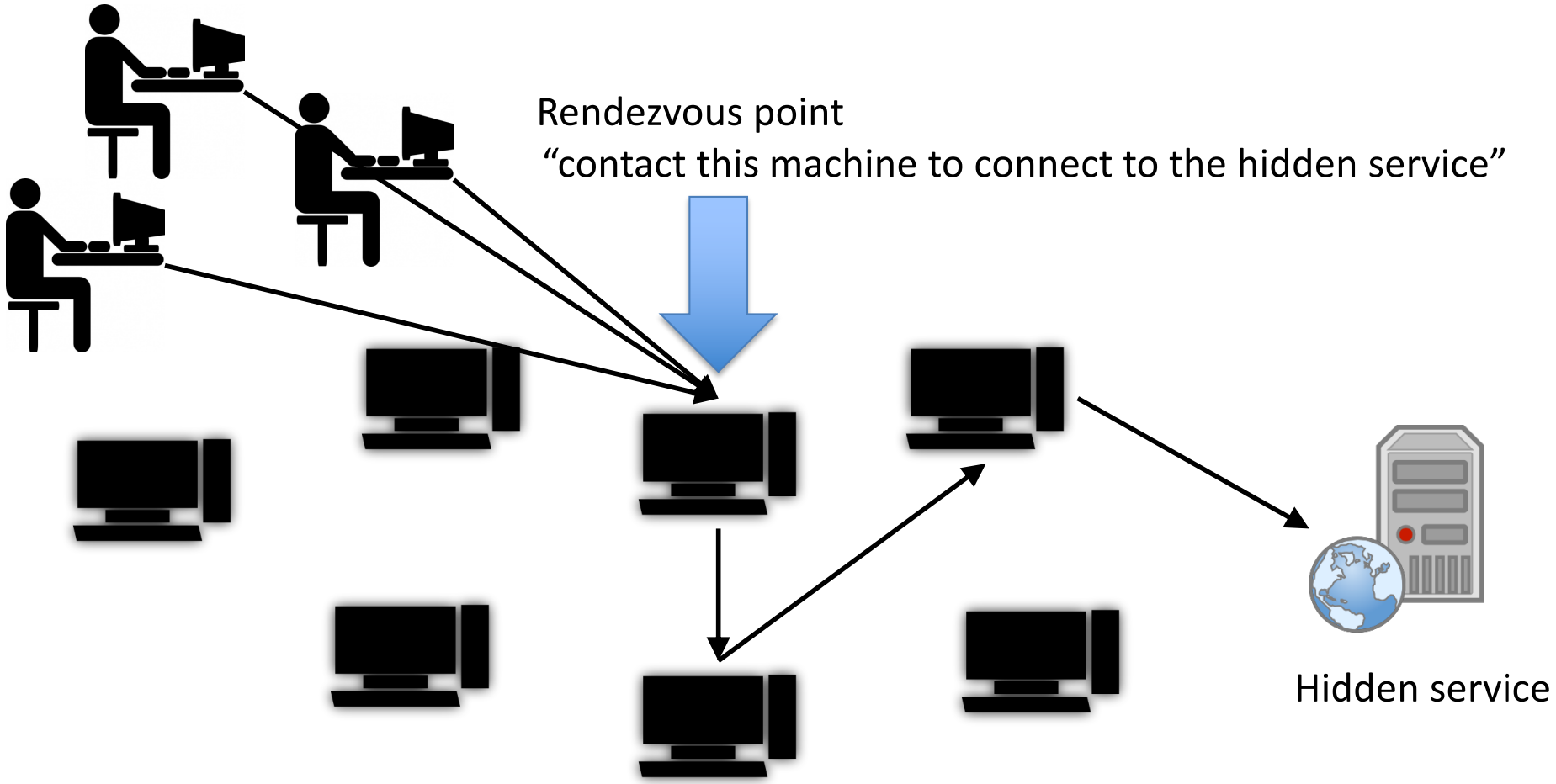
Hidden Services

- Goal: deploy a server on the Internet
  - that anyone can connect to
  - without knowing where it is
    - i.e., without knowing its IP address
  - or who runs it
- -> Resistant to censorship, denial of service, physical attack
  - Network address of the server is hidden, thus can't find (or: very hard to find) the physical server

TOR Hidden Service

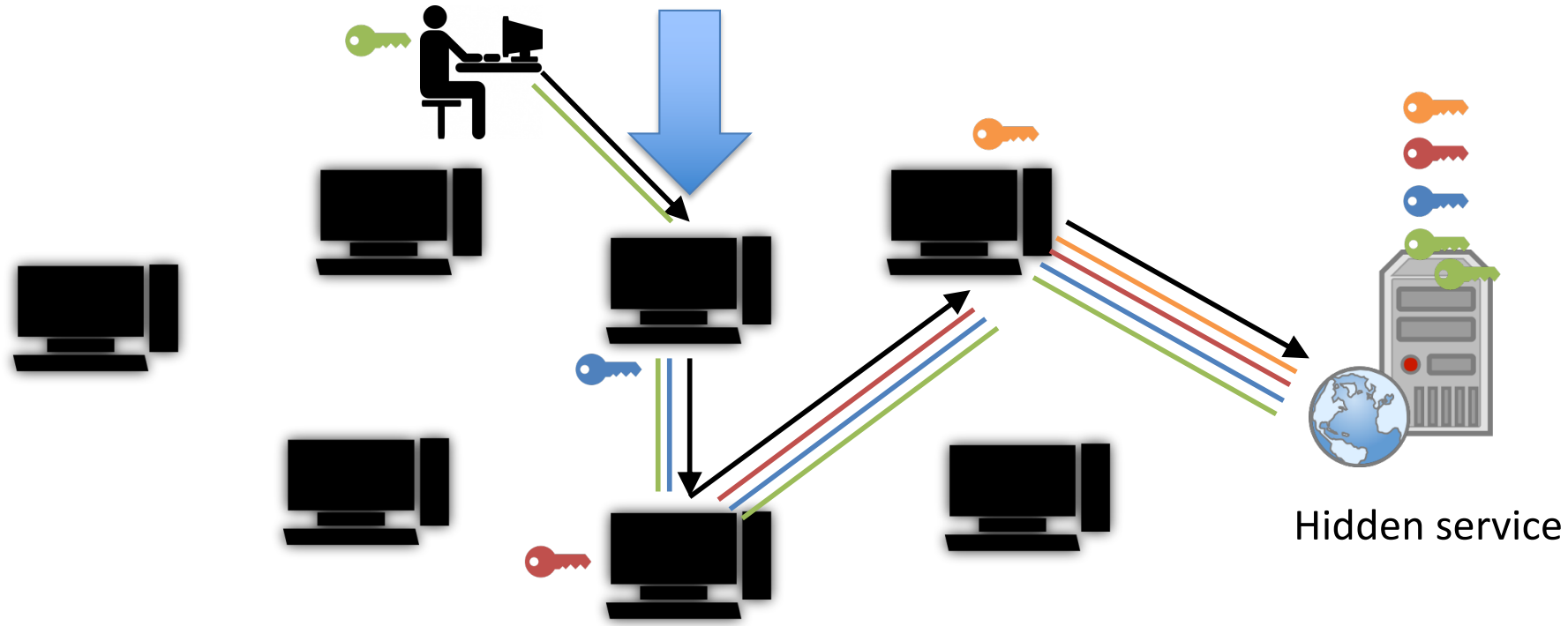


TOR Hidden Service

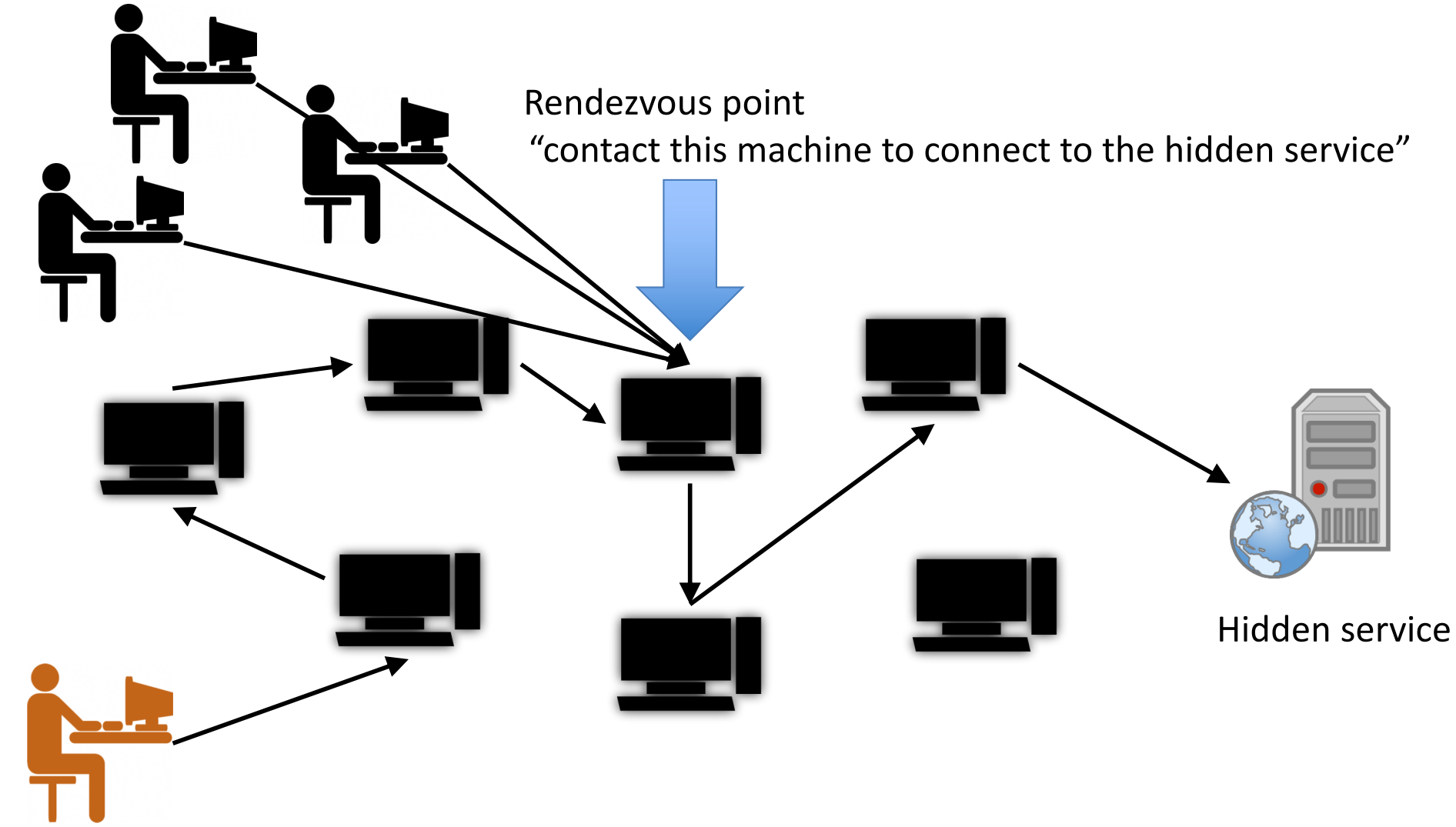


TOR Hidden Service

Layered Encryption?
Who should set it up? The hidden service
Each hop adds / removes one layer of encryption: depends on traffic direction



TOR Hidden Service



How to attack TOR?

- Control relay nodes
  - As many as possible
  - Try to follow traffic from source to destination
- Observe traffic patterns
  - Draw conclusions for each hop

TOR: Ethics

- TOR use cases
  - Discuss health issues or financial matters anonymously
  - Bypass Internet censorship in parts of the world
  - Conceal interaction with gambling, drug, … sites
  - Law enforcement
  - Provide anonymous services
    - Content prone to censorship in parts of the world
    - Gambling, drugs, …
- Does this make TOR a good or bad service? Or both?

“Dark Side”

- Bittorent
- TOR
- Blockchains

From the Dark Side…

White Side

Today and Next Time

- Three “real” systems
  - Google File System
    - File system for Google applications

Today

  - Google’s MapReduce
    - Distributed processing of large data sets
  - Amazon Dynamo
    - Large-scale data storage

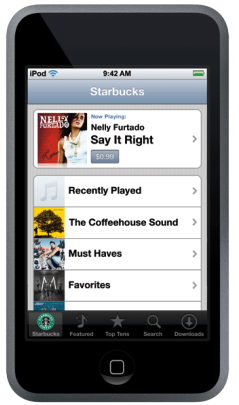
Next Time

Part

Google File System (GFS)

A brief history of Google

=



Google: 1996

4 disk drives

24 GB total storage

A brief history of Google

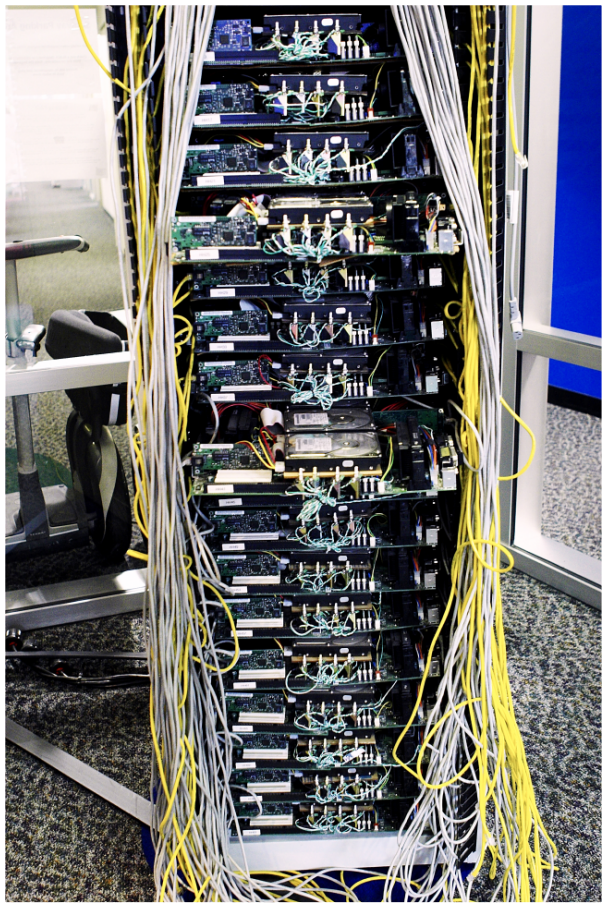
=

Google: 1998

44 disk drives

366 GB total storage

A brief history of Google



Google: 2003

15,000 machines

? PB total storage

A brief history of Google

45 containers x 1000 servers x 36 sites =
~ 1.6 million servers (lower bound)

shipping container

Min 45 containers/data center

Motivation

- At that scale
  - Google needed a good distributed file system
- Approach
  - Redundant storage of massive amounts of data on cheap and unreliable computers

- Why not use an existing file system?
  - Google's problems are different from anyone else's
    - Different workload and design priorities
  - GFS is designed for Google apps and workloads
  - Google apps are designed for GFS

Assumptions

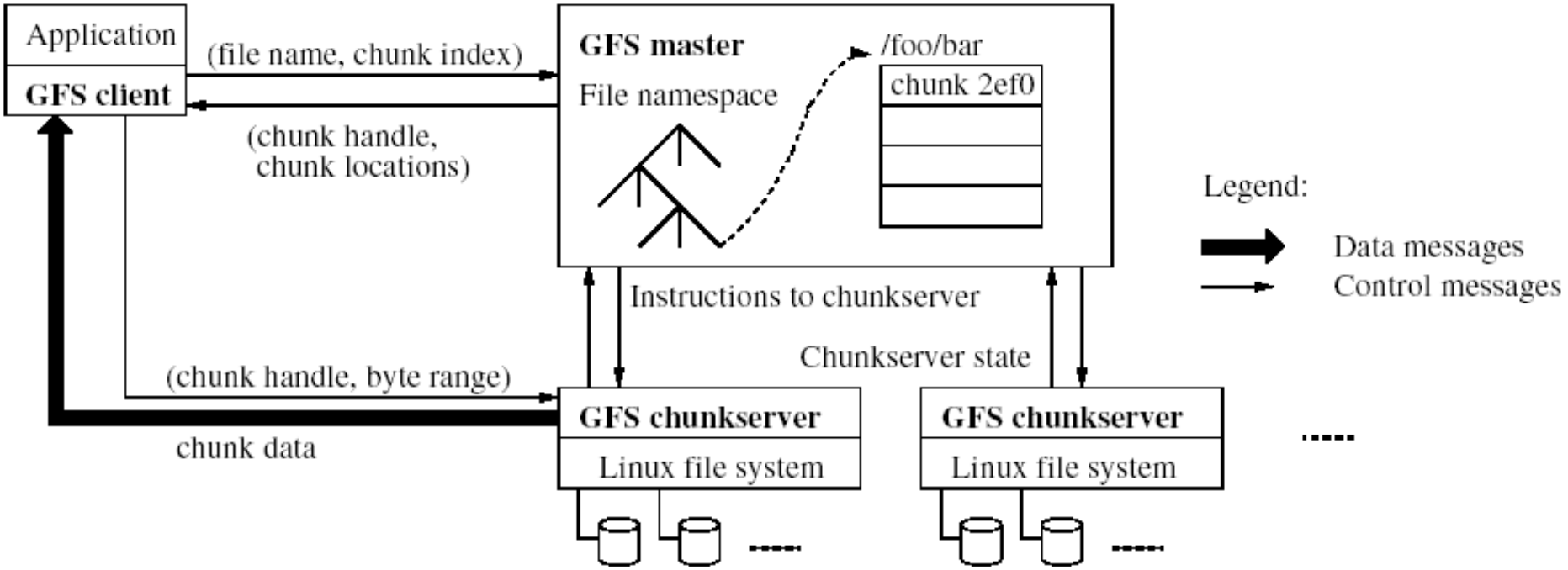
- High component failure rates
  - Inexpensive commodity components fail all the time
- Millions of files
  - multi-GB files typical
- Files are write-once, mostly appended to
  - Perhaps concurrently
- Large streaming reads
- High sustained throughput favoured over low latency

GFS Design Decisions

- Files stored as chunks
  - Fixed size (64MB)
- Reliability through replication
  - Each chunk replicated across 3+ *chunkservers*
- Single master to coordinate access, keep metadata
  - Simple centralized management
- No data caching
  - Little benefit due to large data sets, streaming reads
- Familiar interface, but customize the API
  - Simplify the problem; focus on Google apps
  - Add *snapshot* and *record append* operations

GFS Architecture

- Single master
- Multiple chunk-servers



…Can anyone see a potential weakness in this design?

Single master

- Single master is a:
  - Single point of failure
  - Scalability bottleneck
- GFS solutions:
  - Shadow masters
  - Minimize master involvement
    - never move data through it, use only for metadata
      - and cache metadata at clients
    - large chunk size
    - master delegates authority to primary replicas in data mutations (chunk leases)
- Simple, and good enough!

Metadata (1/2)

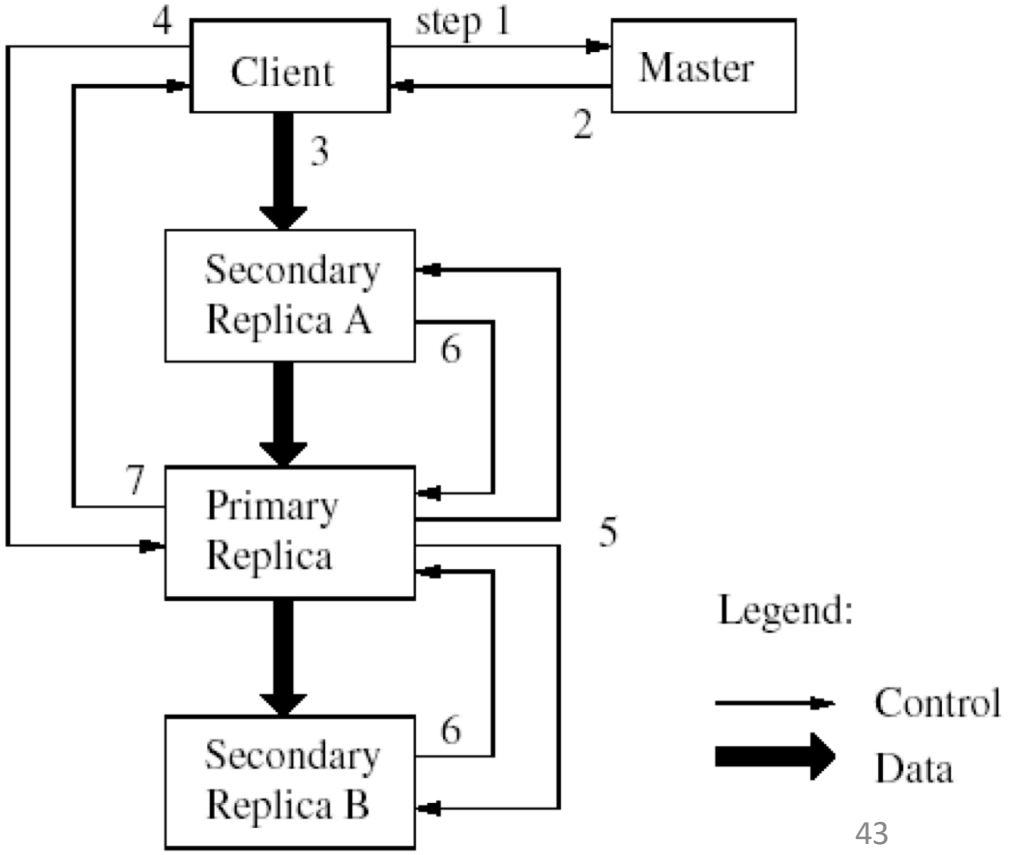
- Global metadata is stored on the master
  - File and chunk namespaces
  - Mapping from files to chunks
  - Locations of each chunk's replicas
- All in memory (64 bytes / chunk)
  - Fast
  - Easily accessible

Metadata (2/2)

- Master has an *operation log* for persistent logging of critical metadata updates
  - persistent on local disk
  - replicated
  - checkpoints for faster recovery

Mutations

- Mutation = write or append
  - must be done for all replicas
- Goal: minimize master involvement
- Lease mechanism:
  - master picks one replica as primary; gives it a “lease” for mutations
  - primary defines a serial order of mutations
  - all replicas follow this order
- Data flow decoupled from control flow



Relaxed Consistency Model

- “Consistent” = all replicas have the same value
- “Defined” = replica reflects the mutation, consistent

- Some properties:
  - concurrent writes leave region consistent, but possibly undefined
    - -> unsure which write was applied first and which second
  - failed writes leave a region inconsistent
- Some work has moved into the applications:
  - e.g., self-validating, self-identifying records

Master's responsibilities (1/2)

- Metadata storage
- Namespace management/locking
- Periodic communication with chunk-servers
  - give instructions, collect state, track cluster health
- Chunk creation, re-replication, rebalancing
  - balance space utilization and access speed
  - spread replicas across racks to reduce correlated failures
  - re-replicate data if redundancy falls below threshold
  - rebalance data to smooth out storage and request load

Master's responsibilities (2/2)

- Garbage Collection
  - simpler, more reliable than traditional file delete
  - master logs the deletion, renames the file to a hidden name
  - lazily garbage collects hidden files
- Stale replica deletion
  - detect "stale" replicas using chunk version numbers

Fault Tolerance

- High availability
  - fast recovery
    - master and chunkservers restartable in a few seconds
  - chunk replication
    - default: 3 replicas.
  - shadow masters
- Data integrity
  - checksum every 64KB block in each chunk
    - In addition to checksums / FEC of the underlying file system
      - Which can have even more frequent checksums

Conclusion

- GFS demonstrates how to support large-scale processing workloads on commodity hardware
  - design to tolerate frequent component failures
  - optimize for huge files that are mostly appended and read
  - feel free to relax and extend FS interface as required
  - go for simple solutions (e.g., single master)

- GFS has met Google's storage needs…
  - it must be good!

GFS Discussion

- Summarize GFS

- olafland.polldaddy.com/s/gfs
  - GFS is highly fault tolerant
  - GFS has a single point of failure
  - Writes in GFS are send to the master node
  - GFS allows any degree of redundancy

olafland.polldaddy.com/s/gfs

GFS Discussion

- olafland.polldaddy.com/s/gfs
  - GFS is highly fault tolerant
    - yes
  - GFS has a single point of failure
    - No: shadow masters
  - Writes in GFS are send to the master node
    - No: directly to the replica
  - GFS allows any degree of redundancy
    - yes

Next Time

- MapReduce
- Dynamo

Questions?

In part, inspired from / based on slides from

- Paul Francis
- Kenneth P. Birman
- Mike Freedman and Tanenbaum
- Hussam Abu-Libdeh
- Steve Schlosser
- Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels
- Jeff Dean
- Alex Moshchuk
- Vamsi Thummala, Prof. Cox
- Scott Shenker
- Ion Stoica
- Vivek Vishnumurthy
- Sukumar Ghosh
- Vitaly Shmatikov