**Incentives build robustness in BitTorrent**
by Bram Cohen

**BitTorrent Protocol Specification**
http://www.bittorrent.org/protocol.html

# BIT TORRENT

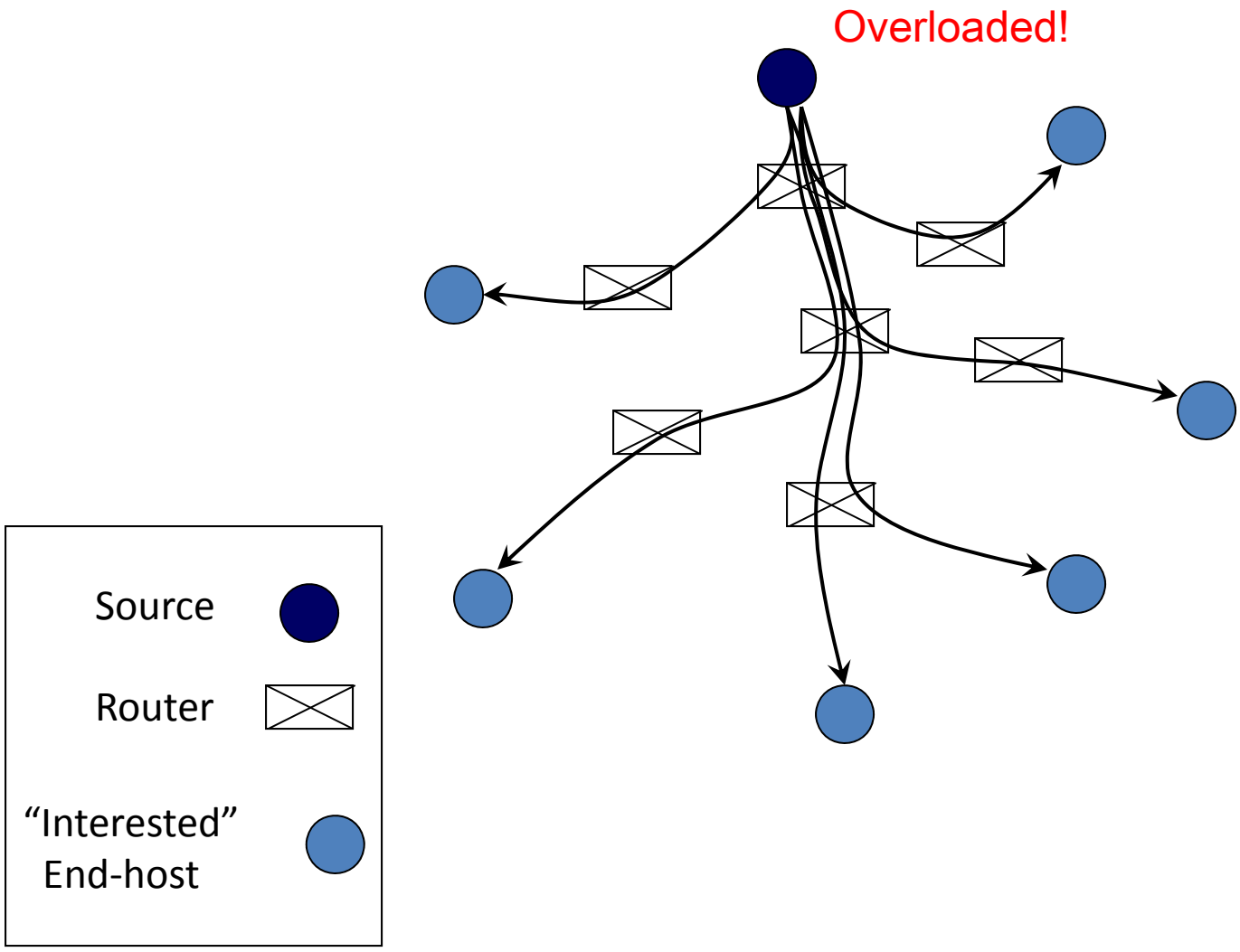
# BitTorrent

- Written by Bram Cohen (in Python) in 2001
- Pull-based, swarming approach (segmented)
  - Each file is split into smaller pieces (& sub-pieces)
  - Peers request desired pieces from neighboring peers
  - Pieces are not downloaded in sequential order
- Encourages contribution by all peers
  - Based on a **tit-for-tat model**

# BitTorrent Use cases

- File-sharing
- *What uses does BitTorrent support?*
  - Downloading (licensed only ☺) movies, music, etc.
  - *And …?*
    - Update distribution among servers at Facebook et al.
    - Distribution of updates and releases (e.g., World of Warcraft -> Blizzard Downloader)
    - …

# Client-server



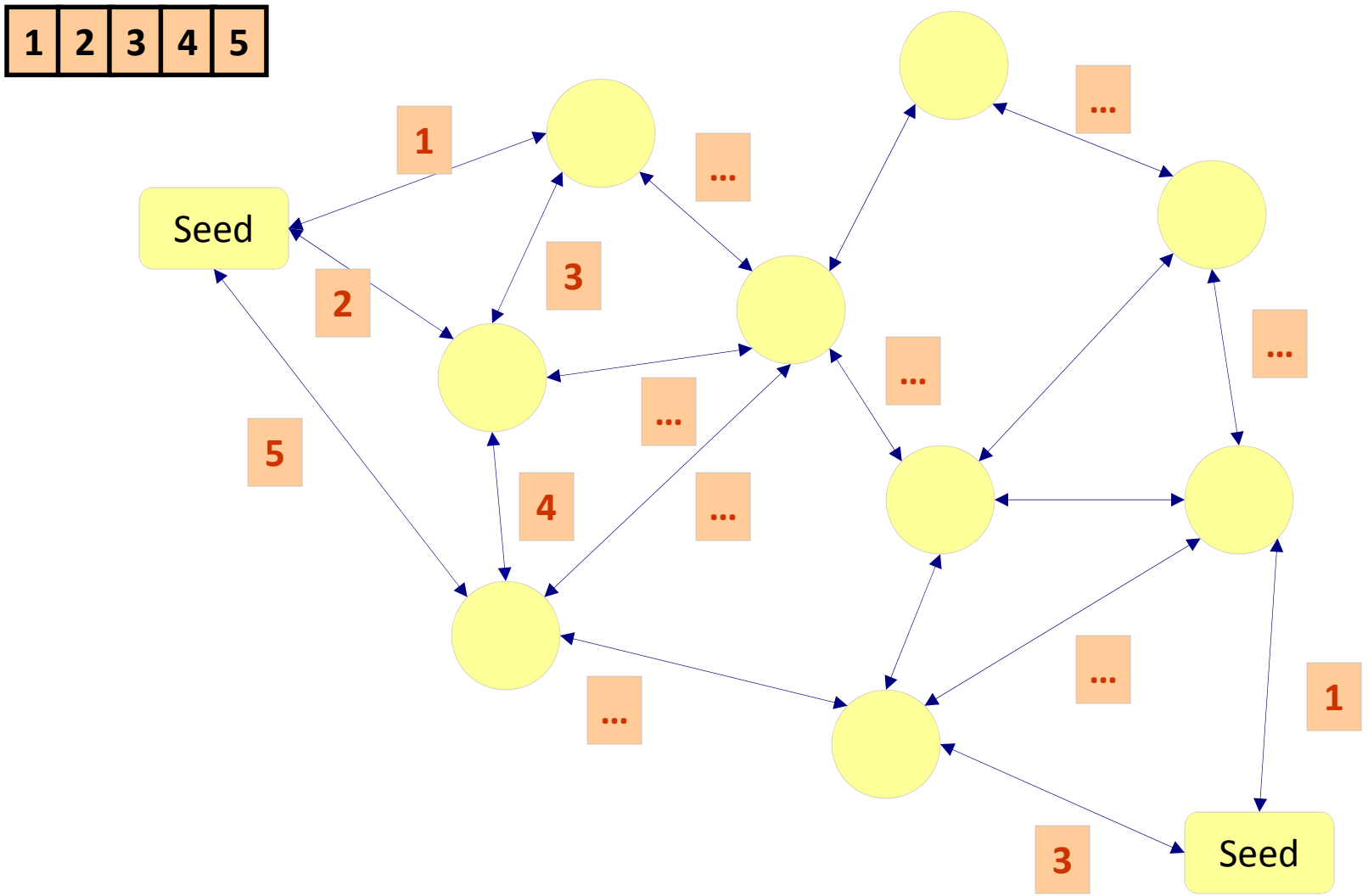
# BitTorrent Swarm

- **Swarm**
  - Set of peers downloading the same file
  - Organized as a randomly connected mesh of peers
- Each peer knows list of pieces downloaded by neighboring peers
- Peer requests pieces it does not own from neighbors

# BitTorrent Terminology

- **Seed**: Peer with the entire file
  - **Original Seed**: First seed for a file
- **Leech**: Peer downloading the file
  - Leech becomes a seed, once file downloaded, if the peer stays online & continues by protocol
- **Sub-piece**: Further subdivision of a piece
  - "Unit for requests" is a sub-piece (16 kB)
  - Peer uploads piece only after assembling it completely

# BitTorrent Swarm



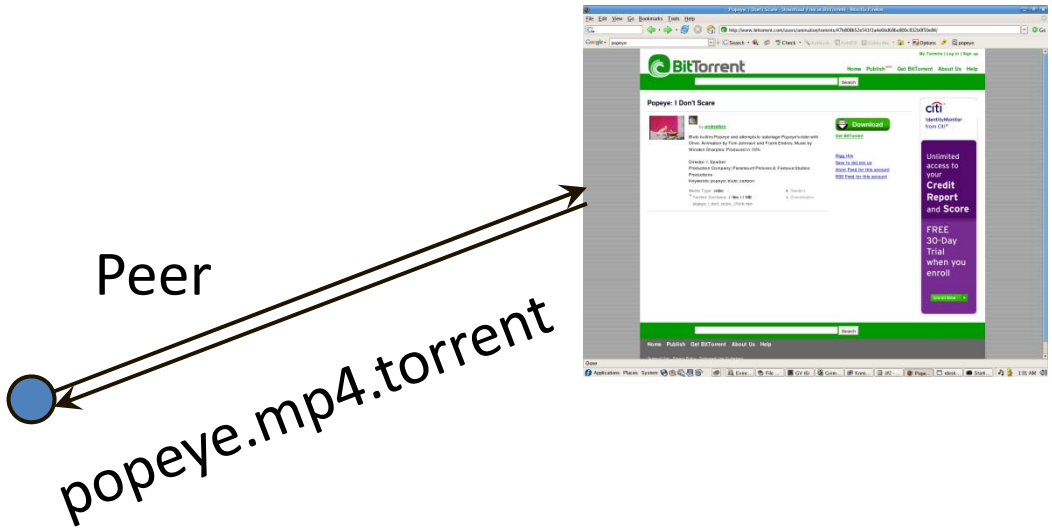

* From Analyzing and Improving BitTorrent by Ashwin R. Bharambe, Cormac Herley and Venkat Padmanabhan

# Entering a Swarm
## for file “popeye.mp4”

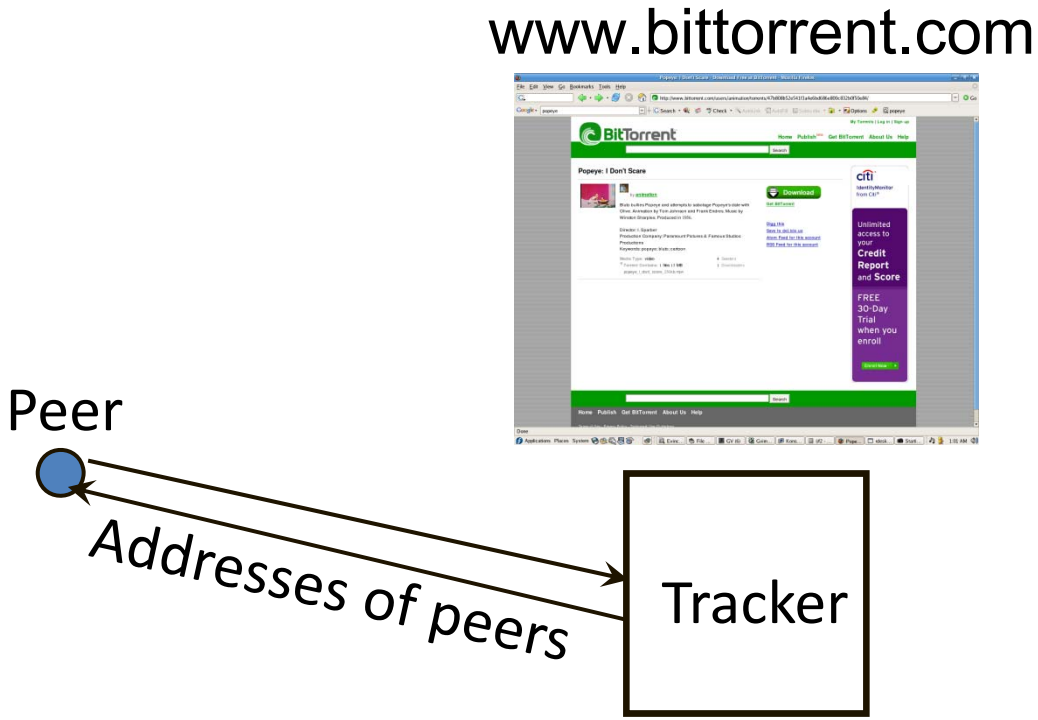
- File **popeye.mp4.torrent** hosted at a (well-known) web server
- The **.torrent** has address of **tracker** for file
- The tracker, which runs on a web server as well, keeps track of all peers downloading file

# Find the Torrent for Desired Content

www.bittorrent.com or
anywhere

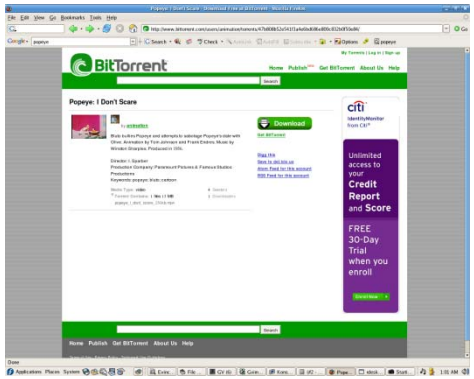


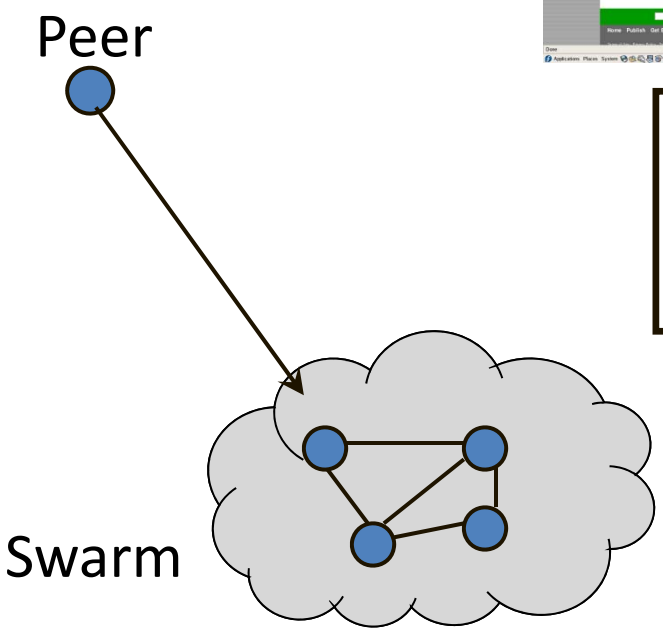
# Contact the Tracker of Retrieved Torrent



# Connect to Available Peers

www.bittorrent.com



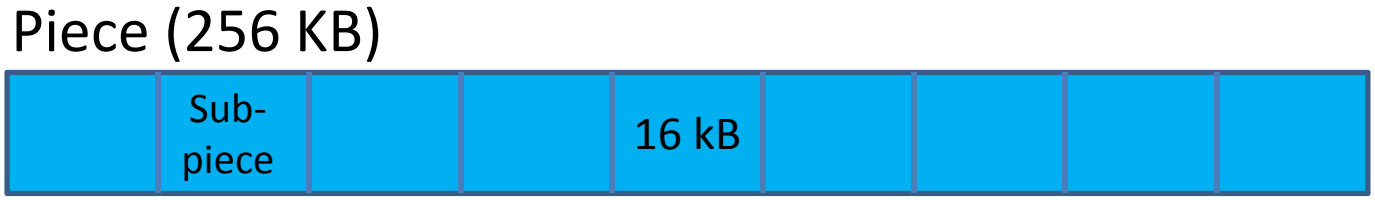



# Contents of .torrent File

- URL of tracker
- Piece length – usually 256 KB
- SHA-1 hashes of each piece in file - ***Why?***

# Download Phases

- **Beginning**: First piece
- **Middle**: Piece 2 to n-x
- **End-game**: Pieces n-x to n



# Middle: Choosing Pieces to Request

- *What is a good strategy?*
  - Most frequent first vs. rarest first?
- **Rarest-first**: Look at all pieces at all neighboring peers and request a piece that's owned by the fewest peers – ***Why?***
  - **Increases diversity** in pieces downloaded
    - Avoids case where a node and each of its peers have exactly the same pieces
    - Increases system-wide throughput
  - Increases **likelihood all pieces still available** even if original seed leaves before any one node has downloaded entire file

# Choosing Pieces to Request: Initial Piece

- First, pick a random piece (**random first policy**)
  - When peer starts to download, request random piece from a peer
  - When first complete piece assembled, **switch to rarest-first**
  - Get first piece to quickly participate in swarm with upload
  - Rare pieces are only available at few peers (slows download down)
- **Strict priority policy**
  - Always **complete download of piece** (sub-pieces) before starting a new piece
  - Getting a complete piece as quickly as possible

# Choosing Pieces to Request: Final Pieces

## End-game mode

- Coupon's collector problem
- Send requests for the final piece to **all peers**
- Upon download of entire piece, cancel request for downloading that piece from other peers
- Speeds up completion of download, otherwise last piece could delay completion of download

- ***Why isn't this done all along?***

# Why BitTorrent took off

- Working implementation (Bram Cohen) with simple well-defined interfaces for publishing content
- Open specification
- Many competitors got sued & shut down (Napster, KaZaA)
- Simple approach
  - Doesn't do "search" per se
  - Users use well-known, trusted sources to locate content

# **Pros** & Cons of BitTorrent

- Proficient in utilizing partially downloaded files
- Discourages “freeloading”
  - By rewarding fastest uploaders
- Encourages diversity through “rarest-first”
  - Extends lifetime of swarm
- Works well for “hot content”

# Pros & **Cons** of BitTorrent

- Assumes all interested peers active at same time
- Performance deteriorates if swarm “cools off”
- Too much overhead to disseminate small files

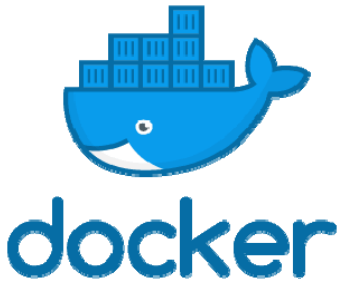
# Pros & Cons of BitTorrent

- Dependence on centralized trackers
  - Single point of failure
    - New nodes can't enter swarm if tracker goes down
    - Simple to design, implement, deploy
  - Today, BitTorrent supports trackerless downloads

- Lack of a search feature
  - Users need to resort to out-of-band search: Well known torrent-hosting sites, plain old web-search

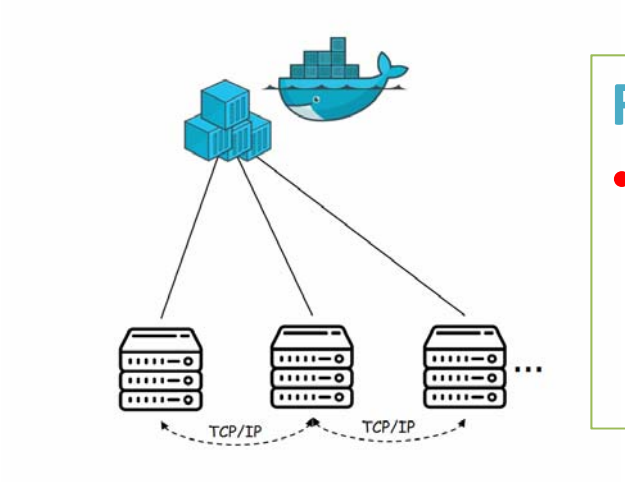
# Docker Image Distribution via BitTorrent

- Docker images are containers that encapsulate applications and systems with full run-time stacks
- Image comprised of minimal file system composed of layers
- When launching a system based on containers, its images must be present on every node.
- Layers are downloaded from **central registry**
- Layers can be downloaded **in parallel**



# Traditional Approach

- Every node downloads image layers from registry




**Fact:**

- 76% of the time spent on container deployment is spent on downloading layers

- Problem
  - High contention at registry
  - Images are pulled entirely from registry to every node
  - Massive network bandwidth overhead and performance bottleneck at the central registry!

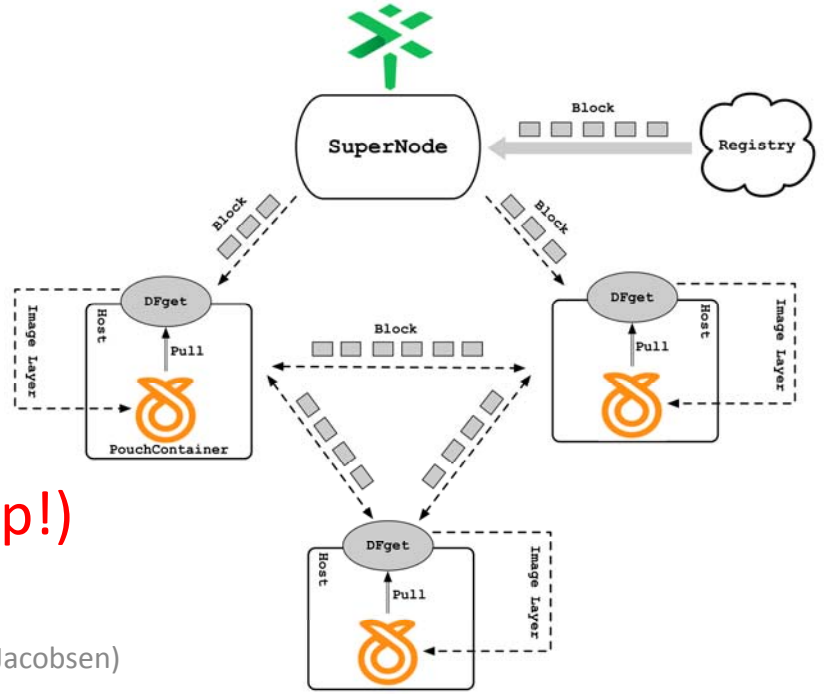
# New Approach

## P2P via BitTorrent (organization-internal)

- Nodes exchange available layers instead of inundating registry
- **Idea**: Chunk layers into small-sized blocks and distribute via BitTorrent (layer*XYZ*.torrent)
- Widely adopted in industry:
  - E.g., Alibaba Dragenfly, Uber Kraken, etc.

https://github.com/dragonflyoss/Dragonfly

https://github.com/uber/kraken




- Drastically improves performance (almost 100x speedup!)