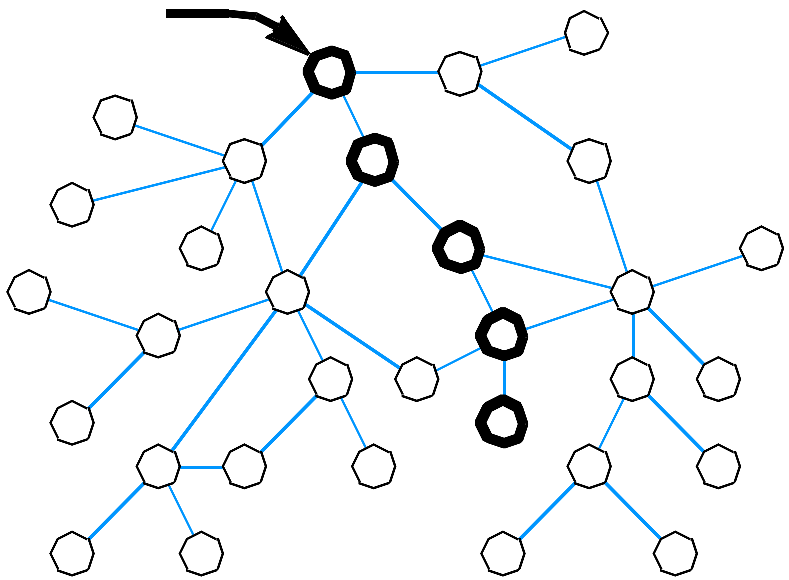
INF5071 – Performance in distributed systems

# Distribution – Part II

October 16, 09

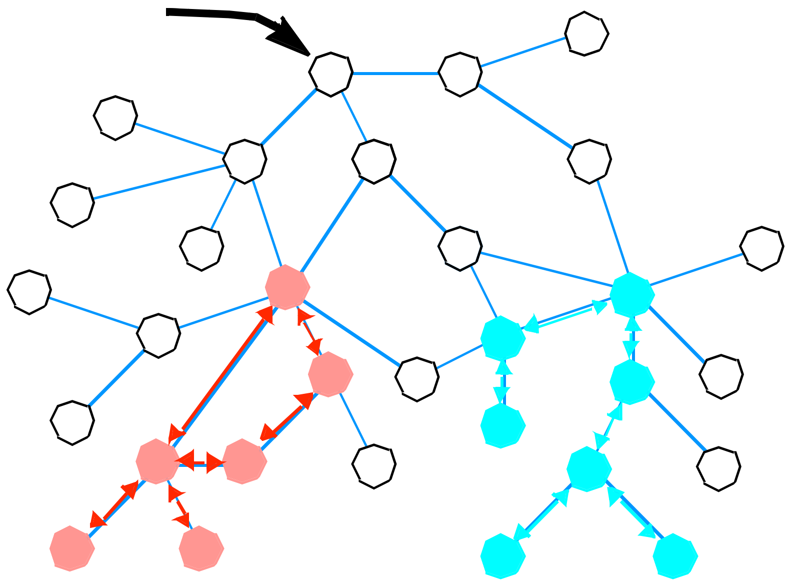
# Type IV – Distribution Systems

- Combine
  - Types I, II or III
  - Network of servers
- Server hierarchy
  - Autonomous servers
  - Cooperative servers
  - Coordinated servers
- "Proxy caches"
  - Not accurate …
  - Cache servers
    - Keep copies on behalf of a remote server
  - Proxy servers
    - Perform actions on behalf of their clients



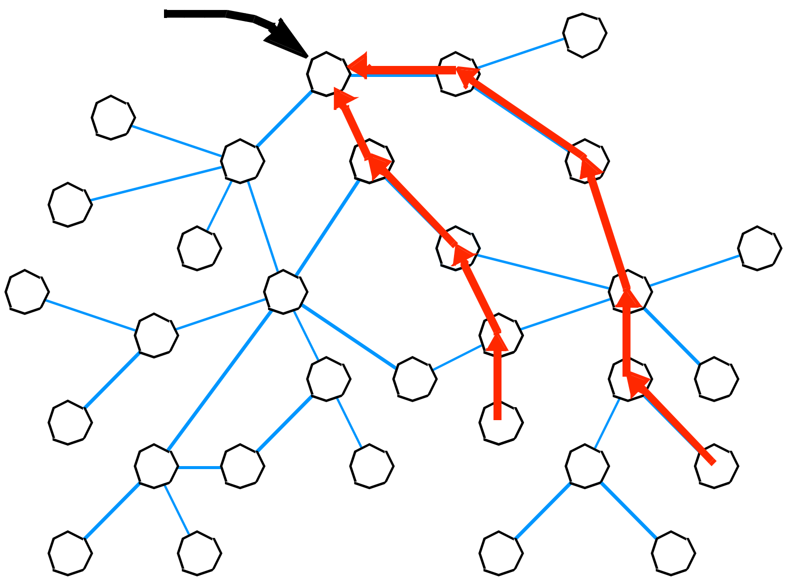
# Type IV – Distribution Systems

- Combine
  - Types I, II or III
  - Network of servers
- Server hierarchy
  - Autonomous servers
  - Cooperative servers
  - Coordinated servers
- "Proxy caches"
  - Not accurate …
  - Cache servers
    - Keep copies on behalf of a remote server
  - Proxy servers
    - Perform actions on behalf of their clients



# Type IV – Distribution Systems

- Combine
  - Types I, II or III
  - Network of servers
- Server hierarchy
  - Autonomous servers
  - Cooperative servers
  - Coordinated servers
- "Proxy caches"
  - Not accurate …
  - Cache servers
    - Keep copies on behalf of a remote server
  - Proxy servers
    - Perform actions on behalf of their clients



# Type IV – Distribution Systems

- Variations
  - Gleaning
    - Autonomous, coordinated possible
    - In komssys
  - Proxy prefix caching
    - Coordinated, autonomous possible
    - In Blue Coat (which was formerly Cacheflow, which was formerly Entera)
  - Periodic multicasting with pre-storage
    - Coordinated
    - The theoretical optimum

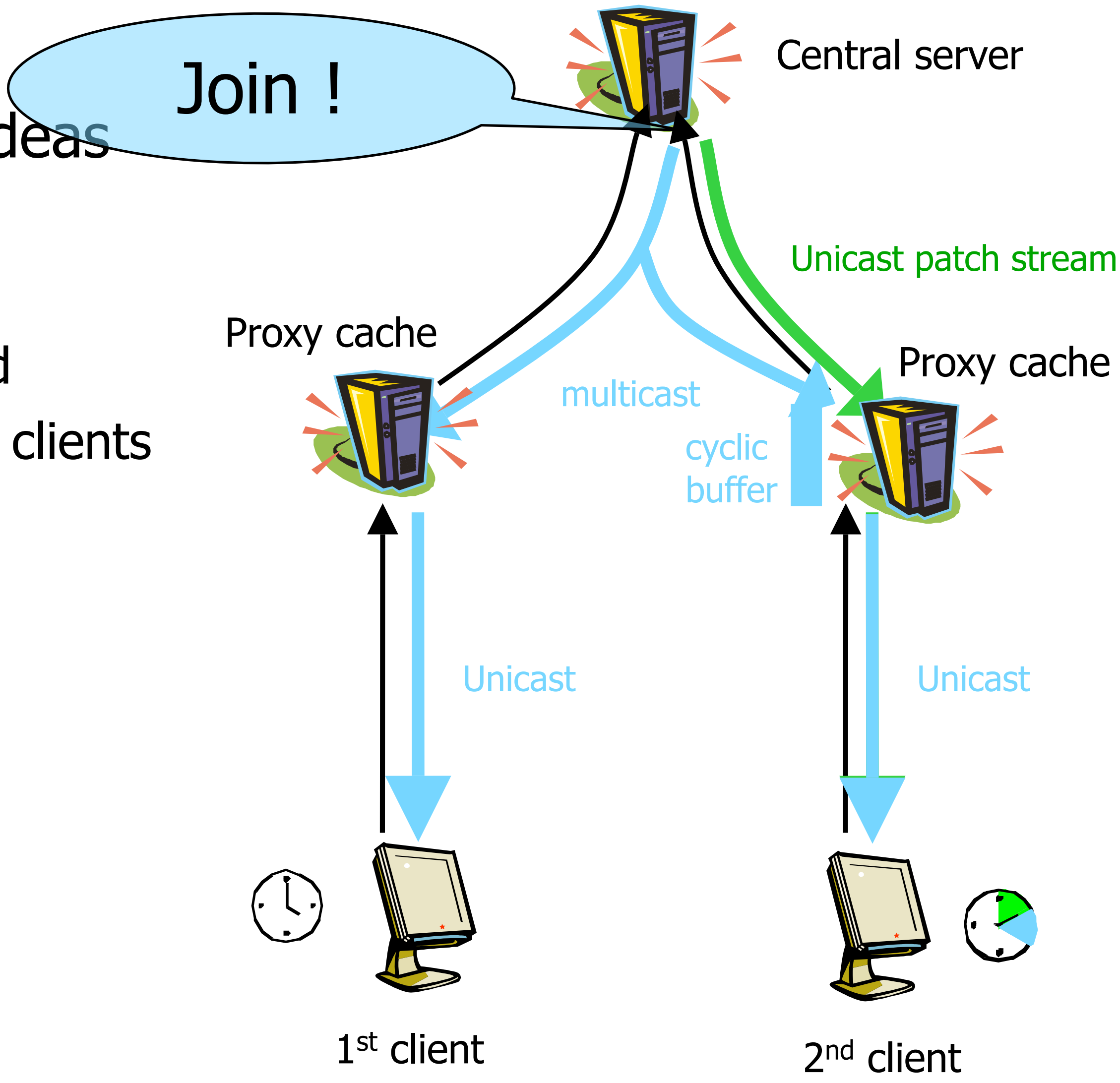
# Gleaning

- Webster's Dictionary: from Late Latin glennare, of Celtic origin
  1. to gather grain or other produce left by reapers
  2. to gather information or material bit by bit

- Combine patching with caching ideas
  - non-conflicting benefits of caching and patching
- Caching
  - reduce number of end-to-end transmissions
  - distribute service access points
  - no single point of failure
  - true on-demand capabilities
- Patching
  - shorten average streaming time per client
  - true on-demand capabilities

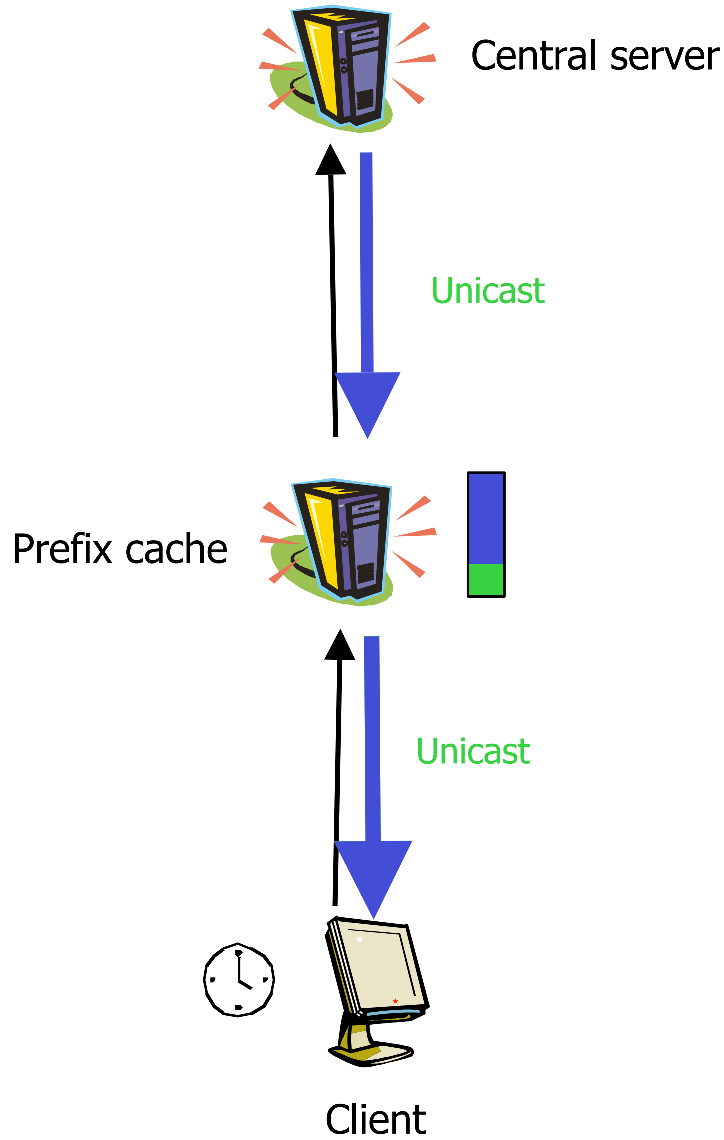
# Gleaning

- Combines
  Patching & Caching ideas
  - Wide-area scalable
  - Reduced server load
  - Reduced network load
  - Can support standard clients



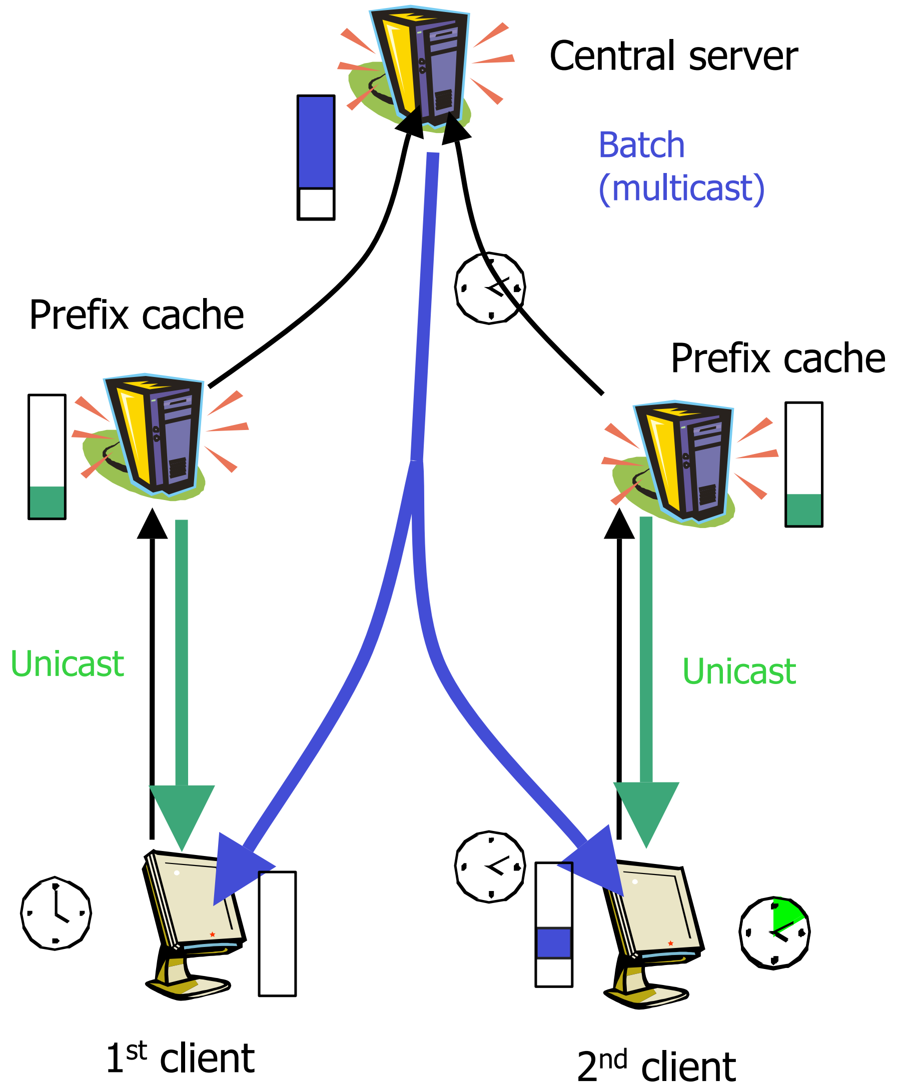
# Proxy Prefix Caching

- Split movie
  - Prefix
  - Suffix
- Operation
  - Store prefix in prefix cache
    - Coordination necessary!
  - On demand
    - Deliver prefix immediately
    - Prefetch suffix from central server
- Goal
  - Reduce startup latency
  - Hide bandwidth limitations, delay and/or jitter in backbone
  - Reduce load in backbone



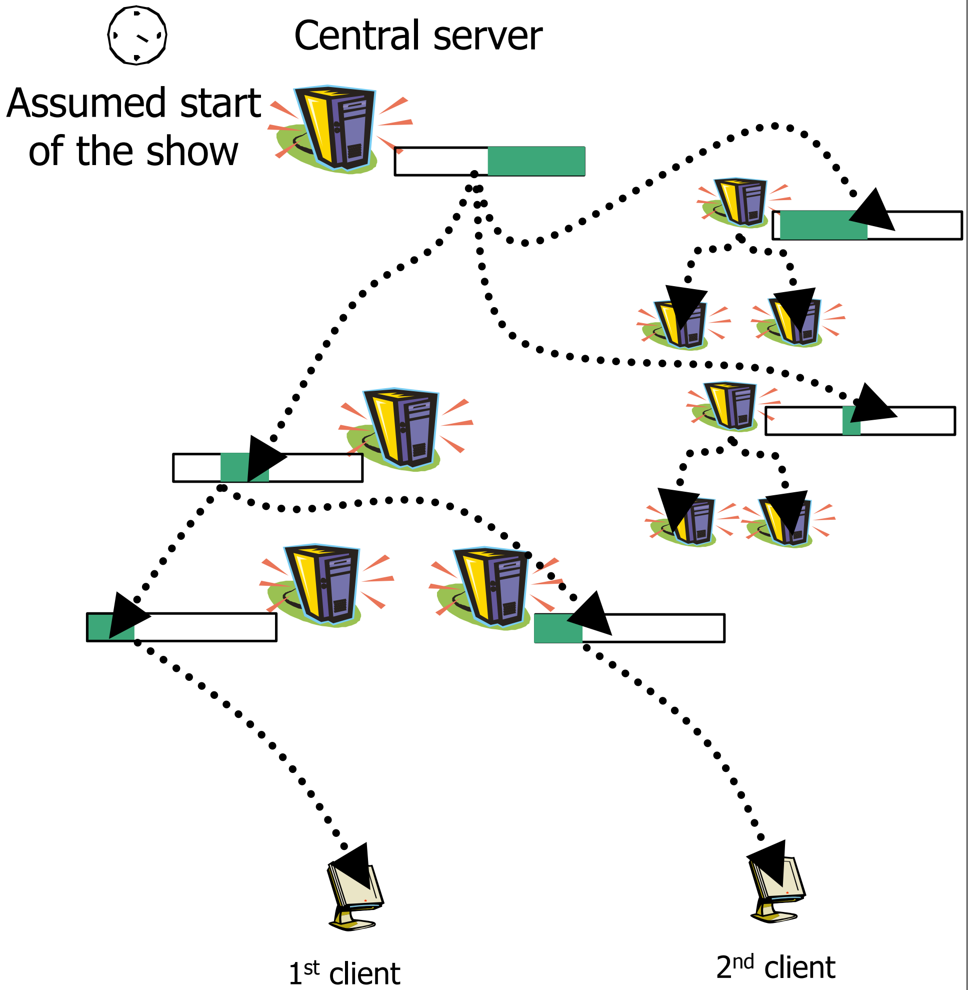
# MCache

- One of several Prefix Caching variations
- Combines Batching and Prefix Caching
  - Can be optimized per movie
    - server bandwidth
    - network bandwidth
    - cache space
  - Uses multicast
  - Needs non-standard clients



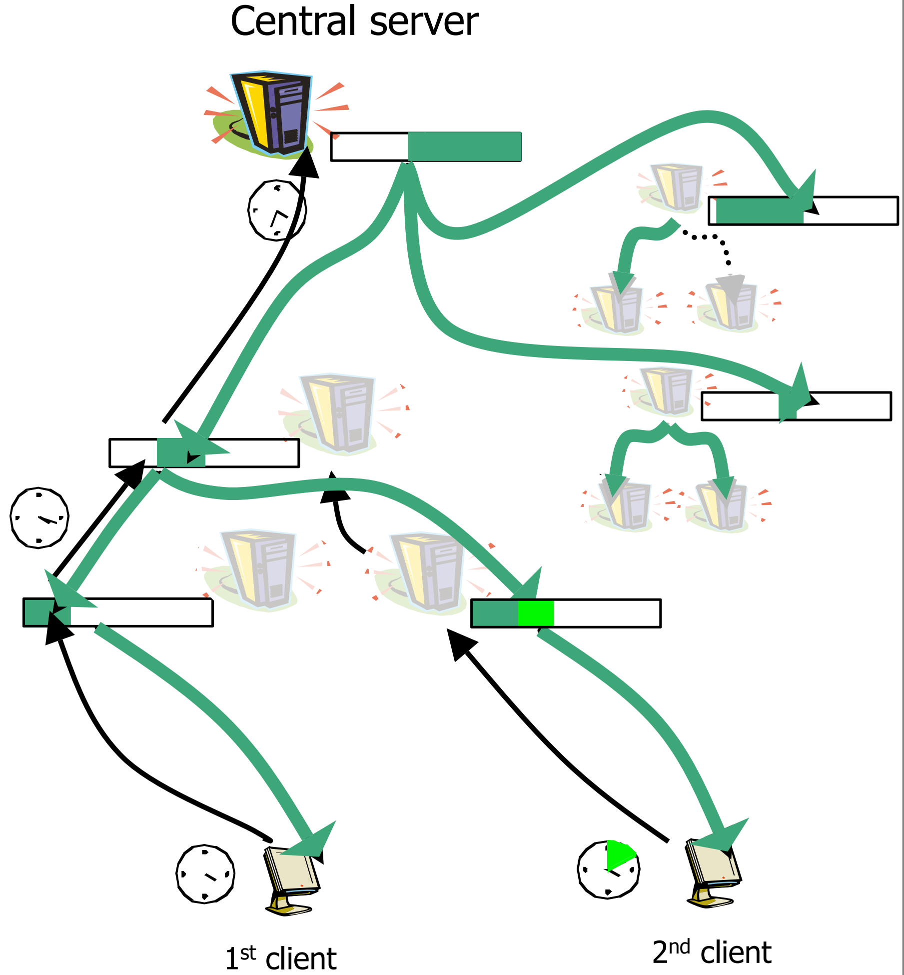
# Periodic Multicasting with Pre–Storage

- Optimize storage and network
  - Wide-area scalable
  - Minimal server load achievable
  - Reduced network load
  - Can support standard clients
- Specials
  - Can optimize network load per subtree
- Negative
  - Bad error behaviour



# Periodic Multicasting with Pre–Storage

- Optimize storage and network
  - Wide-area scalable
  - Minimal server load achievable
  - Reduced network load
  - Can support standard clients
- Specials
  - Can optimize network load per subtree
- Negative
  - Bad error behaviour



# Type IV – Distribution Systems

- Autonomous servers
  - Requires decision making on each proxy
  - Some content must be discarded
  - Caching strategies

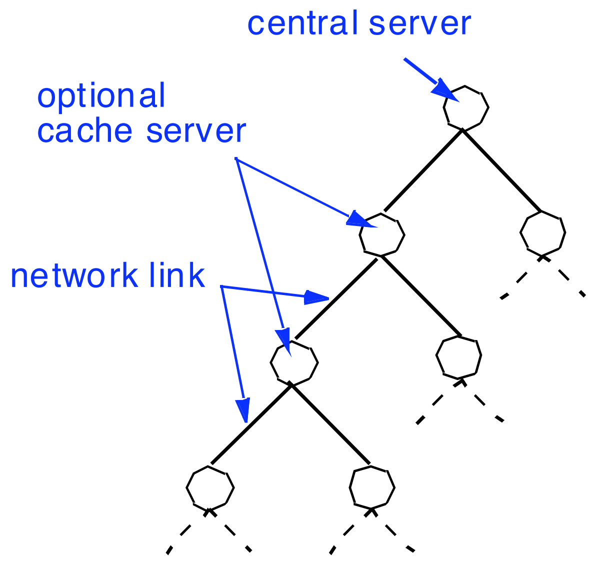
- Coordinated servers
  - Requires central decision making
  - Global optimization of the system

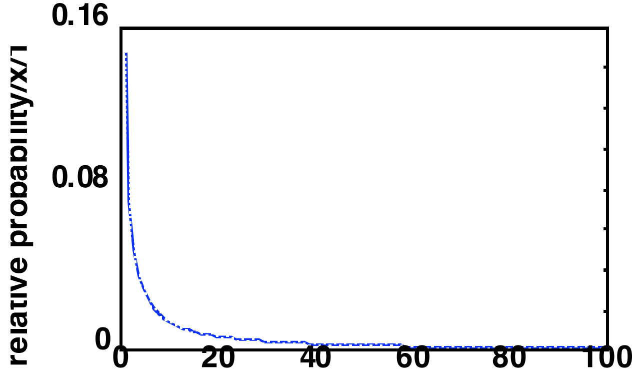
- Cooperative servers
  - No quantitative research yet

# Autonomous servers

# Simulation

- Binary tree model allows
  - Allows analytical comparison of
    - Caching
    - Patching
    - Gleaning
- Considering
  - optimal cache placement per movie
  - basic server cost
  - per-stream costs of storage, interface card, network link
  - movie popularity according to Zipf distribution



# Simulation

- Example
  - 500 different movies
  - 220 concurrent active users
  - basic server: $25000
  - interface cost: $100/stream
  - network link cost: $350/stream
  - storage cost: $1000/stream
- Analytical comparison
  - 👍 demonstrates potential of the approach
  - 👎 very simplified

| | | |
|---|---|---|
| Caching | Caching<br>Unicast transmission | 4664 Mio $ |
| λ-Patching | No caching<br>Client side buffer<br>Multicast | 375 Mio $ |
| Gleaning | Caching<br>Proxy client buffer<br>Multicast | 276 Mio $ |

# Caching Strategies

- FIFO: First-in-first-out
  - Remove the oldest object in the cache in favor of new objects
- LRU: Least recently used strategy
  - Maintain a list of objects
  - Move to head of the list whenever accessed
  - Remove the tail of the list in favor of new objects
- LFU: Least frequently used
  - Maintain a list distance between last two requests per object
  - Distance can be time or number of requests to other objects
  - Sort list: shortest distance first
  - Remove the tail of the list in favor of new objects

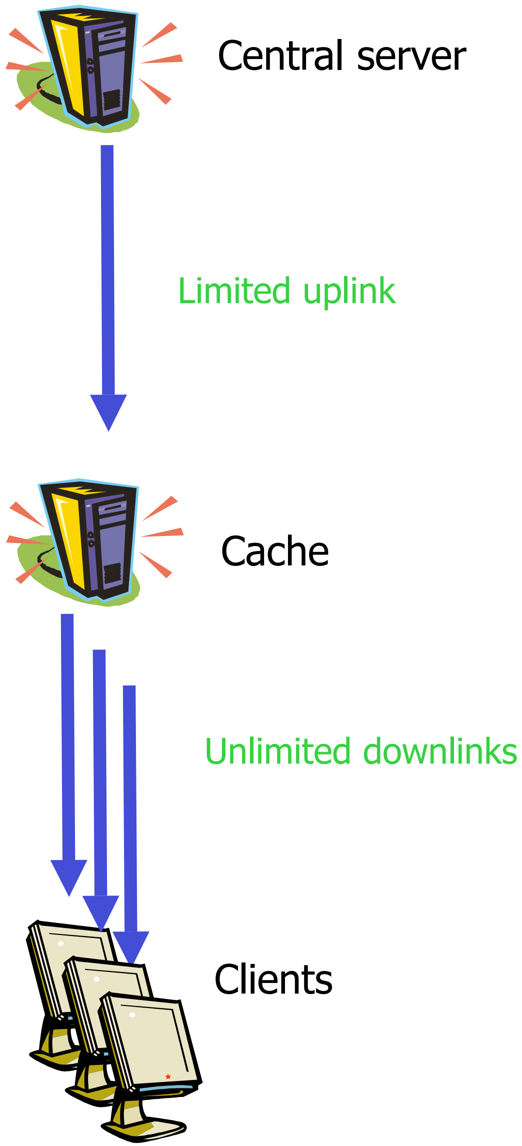
# Caching Strategies

- Considerations
  - limited uplink bandwidth
    - quickly exhausted
    - performance degrades immediately
      when working set is too large for storage space
  - conditional overwrite strategies
    - can be highly efficient
- ECT: Eternal, Conditional, Temporal

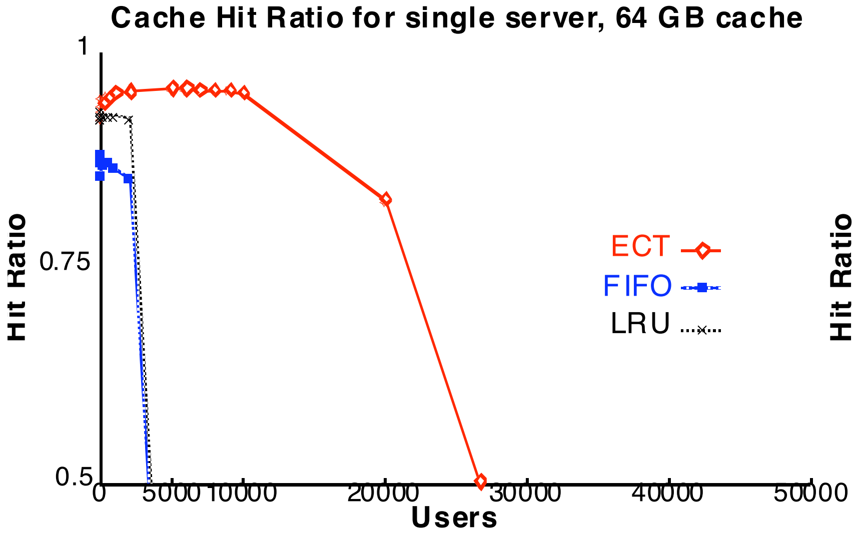
| LFU | | ECT |
|---|---|---|
| Forget object statistics when removed | → | Remember object statistics forever |
| Cache all requested objects | → | Compare requested object and replacement candidate |
| Log # requests or time between hits | → | Log times between hits |

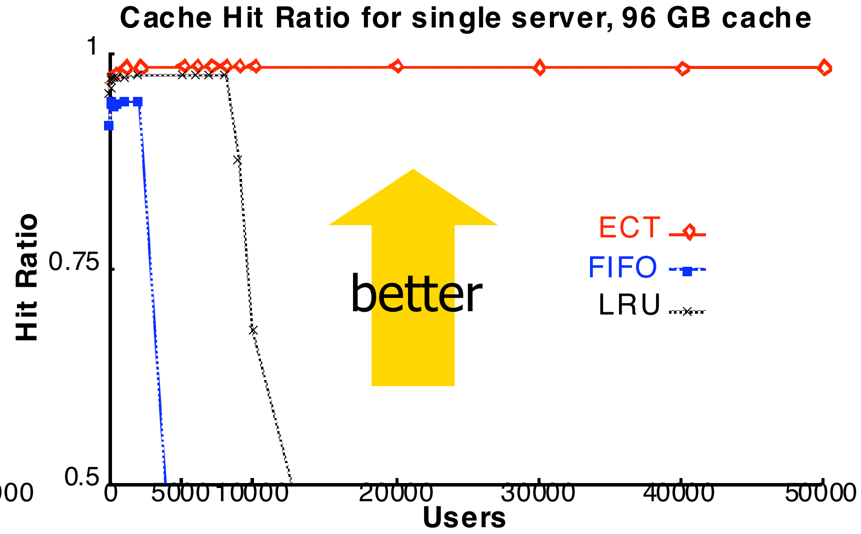
# Simulation

- Movies
  - 500 movies
  - Zipf-distributed popularity
  - 1.5 MBit/s
  - 5400 sec
  - File size ~7.9 GB



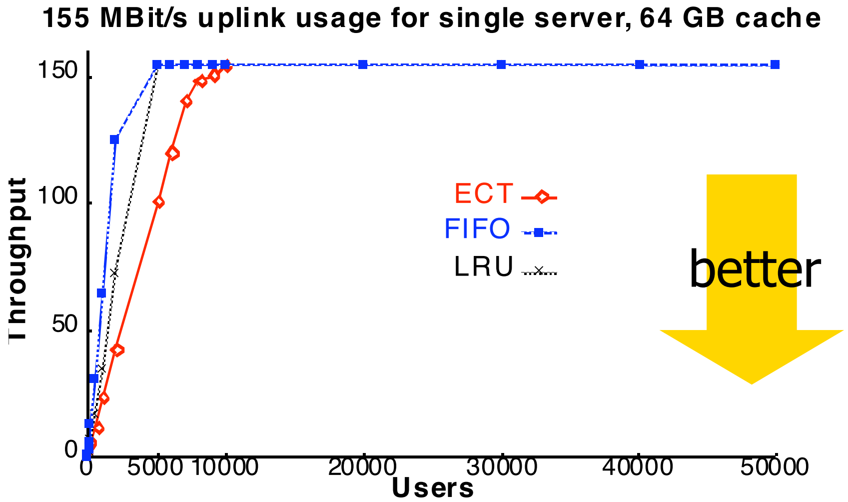
# Effects of caching strategies on user hit rates

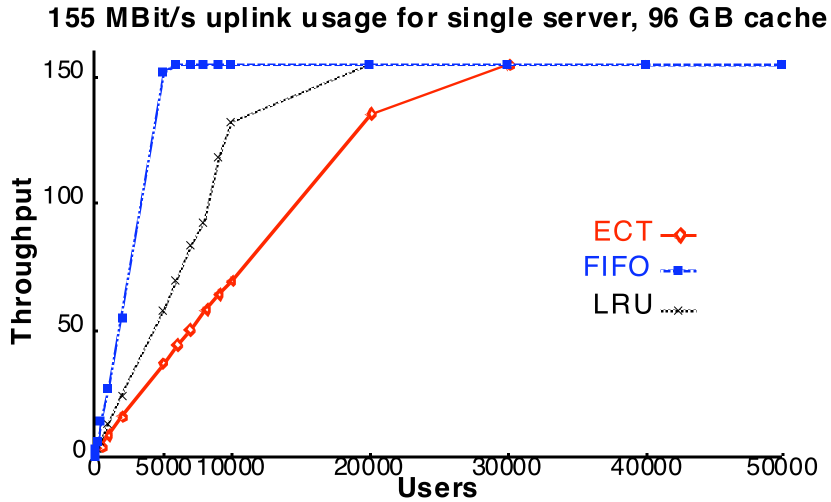



- Hit ratio
  - dumb strategies (almost) do not profit from cache size increases
  - intelligent strategies profit hugely from cache size increases
  - strategies that use conditional overwrite outperform other strategies massively
    - doesn't have to be ECT

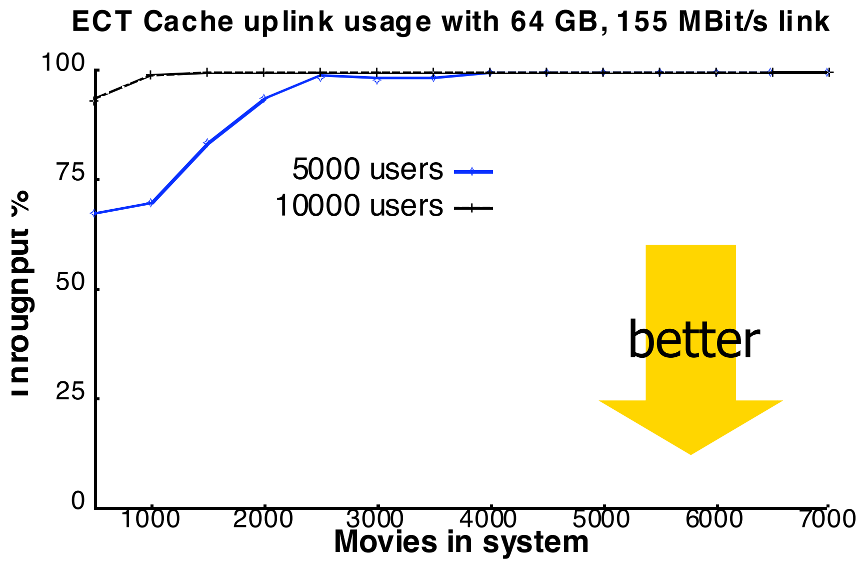
# Effects of caching strategies on throughput

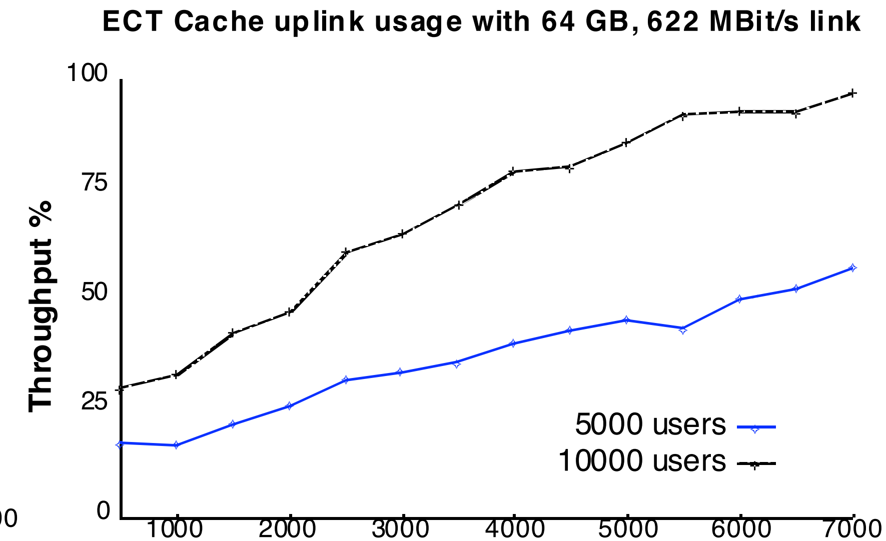



- Uplink usage
  - profits from small cache increase greatly ...... if there is a strategy
  - conditional overwrite reduces uplink usage

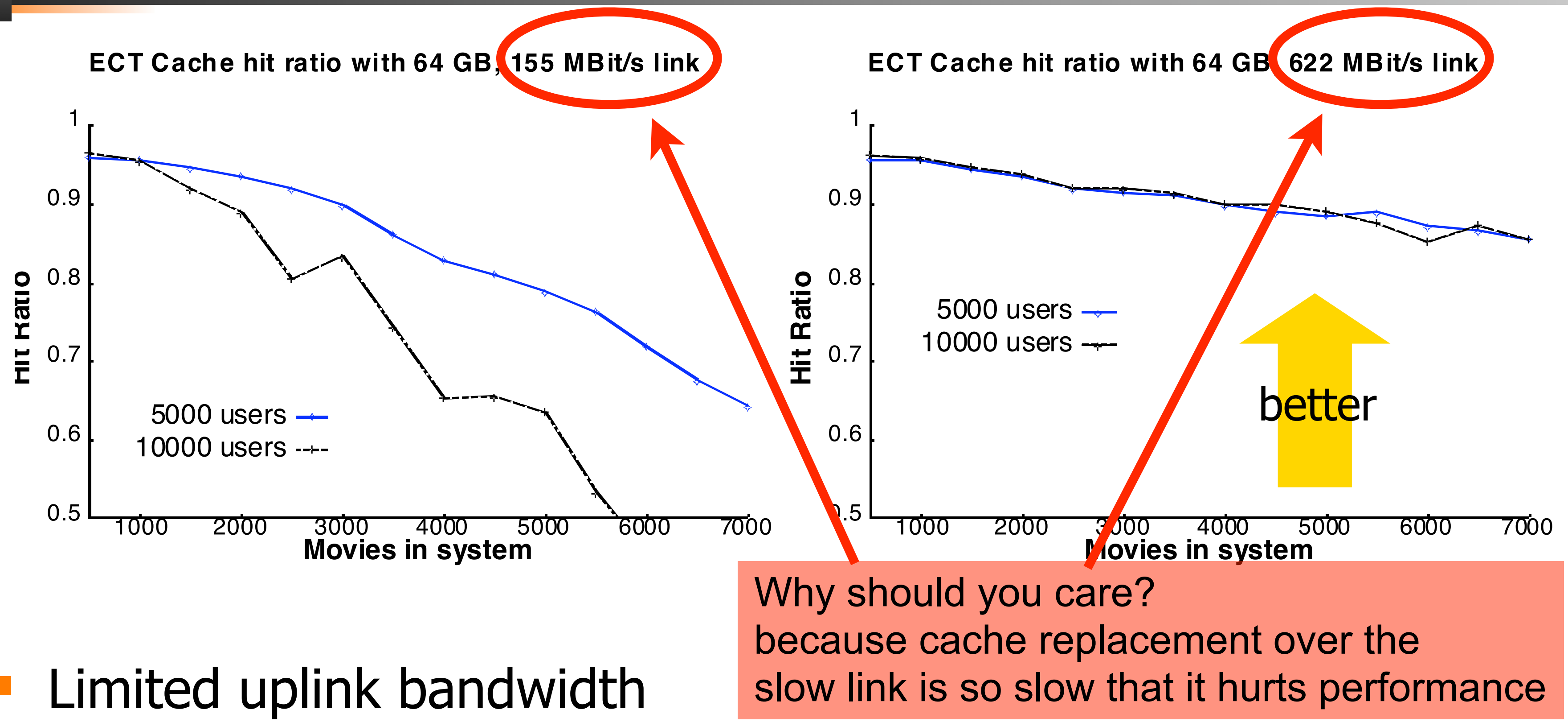
# Effects of number of movies on uplink usage



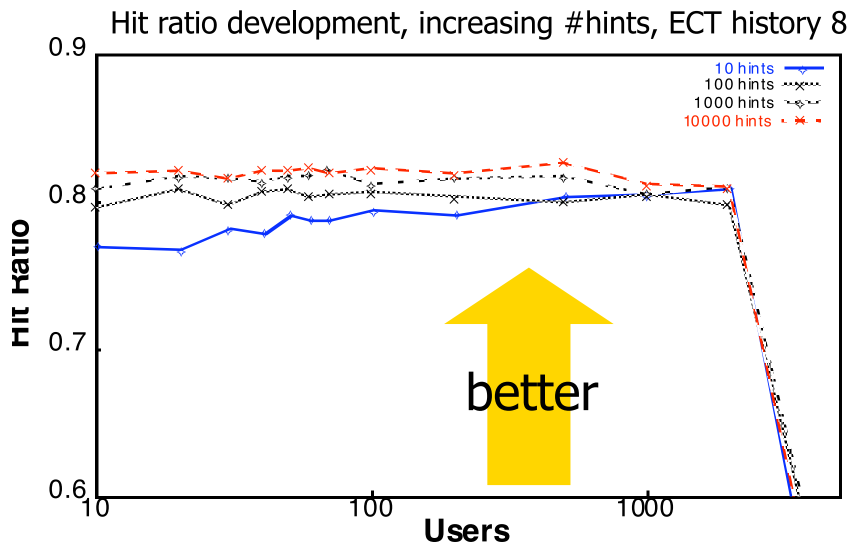

- In spite of 99% hit rates
  - Increasing the number of users will congest the uplink
  - Note
    - scheduling techniques provide no savings on low-popularity movies
    - identical to unicast scenario with minimally larger caches

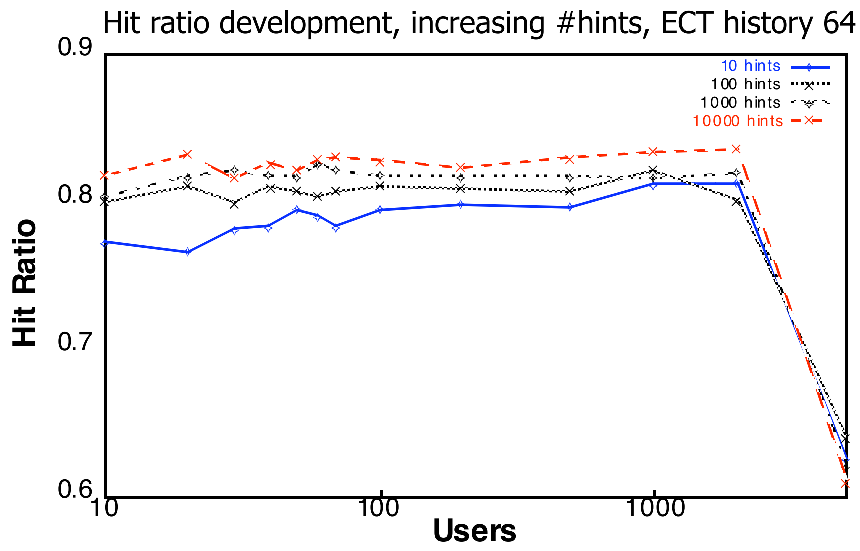
# Effects of number of movies on hit ratio




- Limited uplink bandwidth
  - Prevents the exchange of titles with medium popularity
  - Unproportional drop of efficiency for more users
  - Strategy can not recognize medium popularity titles

# Hint–based Caching




- Idea
  - Caches consider requests to neighbor caches in their removal decisions
- Conclusion
  - Instability due to uplink congestion can not be prevented
  - Advantage exists and is logarithmic as expected
    - Larger hint numbers maintain the advantage to the point of instability
  - Intensity of instability is due to ECT problem
    - ECT inherits IRG (inter reference gap) drawback of fixed–size histograms

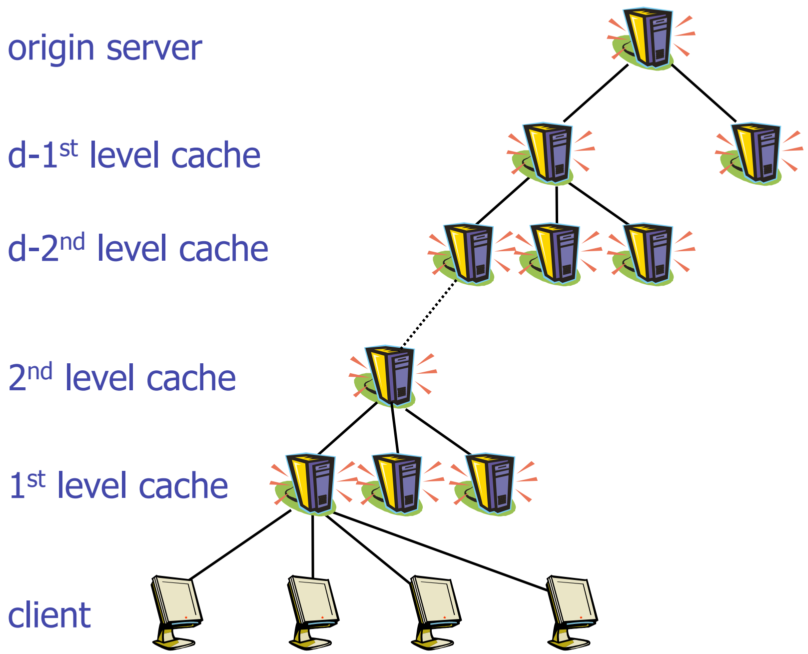
# Simulation: Summary

- High relevance of population sizes
  - complex strategies require large customer bases
- Efficiency of small caches
  - 90:10 rule–of–thumb reasonable
  - unlike web caching
- Efficiency of distribution mechanisms
  - considerable bandwidth savings for uncached titles
- Effects of removal strategies
  - relevance of conditional overwrite
  - unlike web caching, paging, swapping, ...
- Irrelevance of popularity changes on short timescales
  - few cache updates
    compared to many direct deliveries

# Coordinated servers

# Distribution Architectures

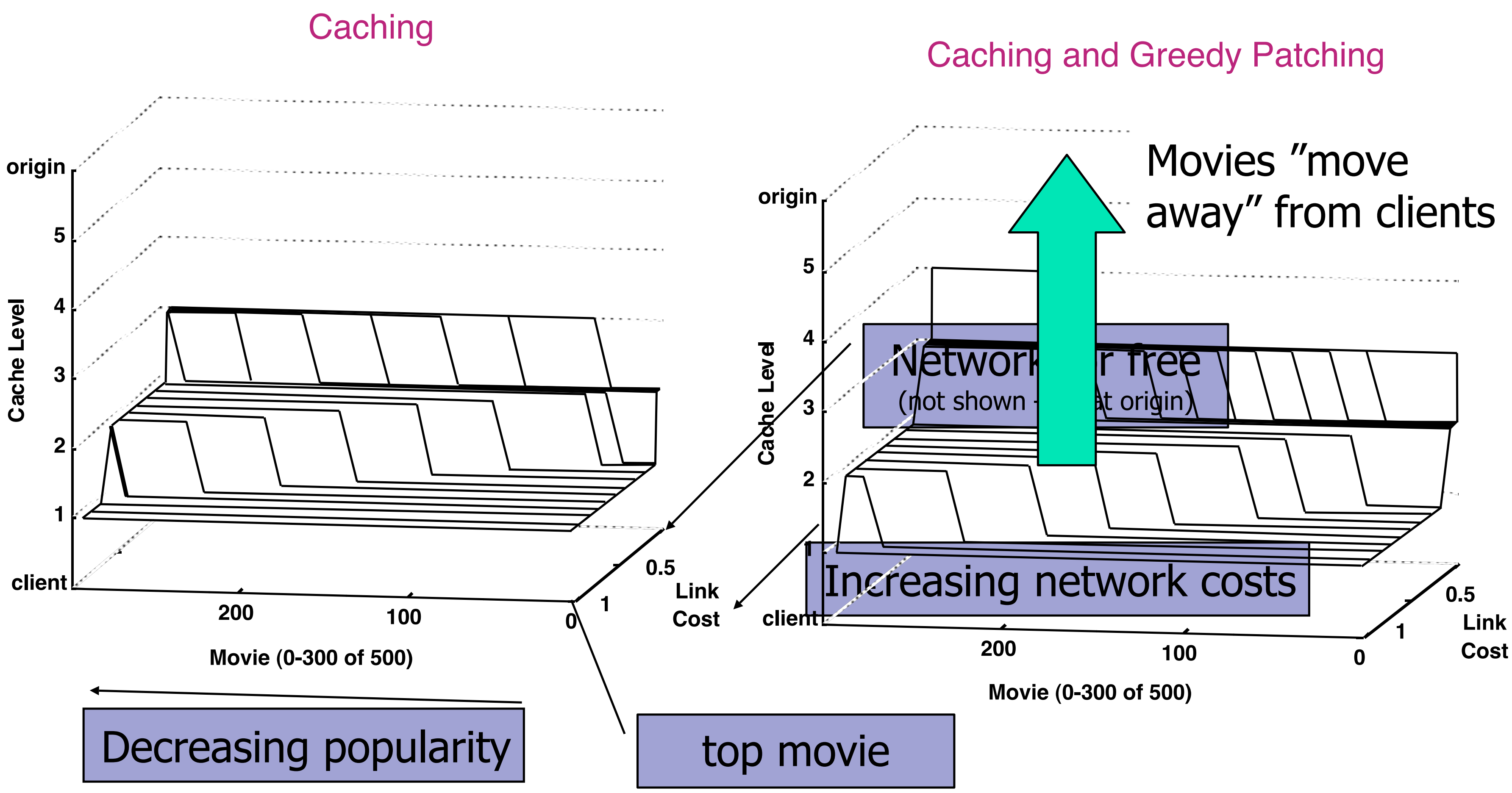
- Combined optimization
  - Scheduling algorithm
  - Proxy placement and dimensioning



# Distribution Architectures

- Combined optimization
  - Scheduling algorithm
  - Proxy placement and dimensioning
- No problems with simple scheduling mechanisms
- Examples
  - Caching with unicast communication
  - Caching with greedy patching
    - Patching window in greedy patching is the movie length

# Distribution Architectures

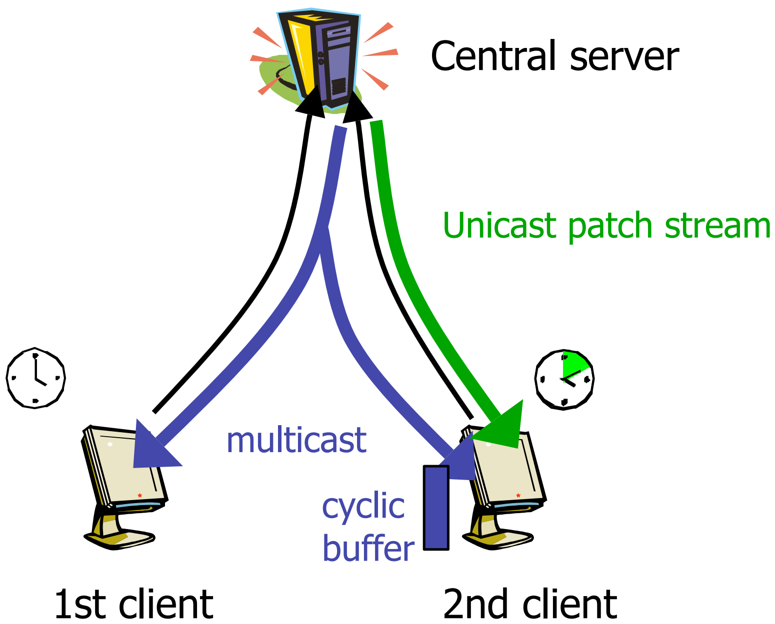


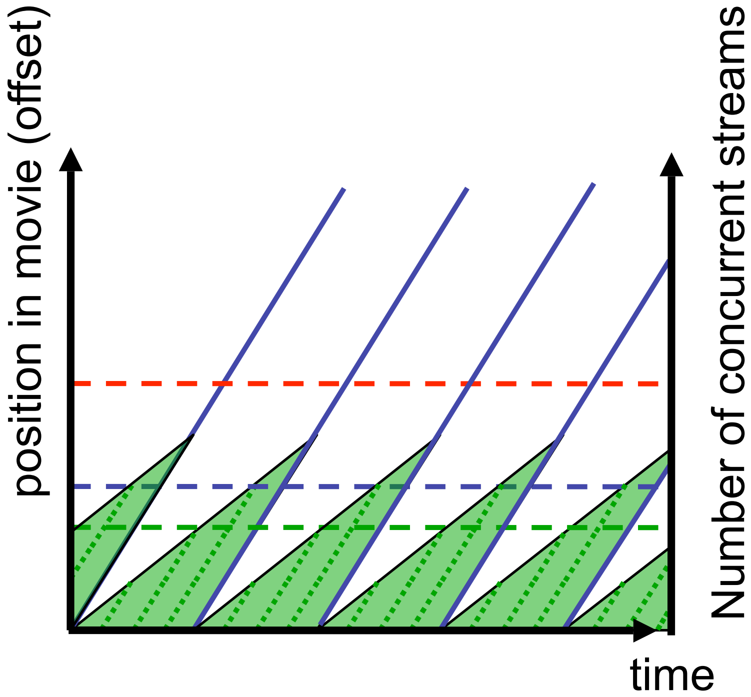
# Distribution Architectures

- Combined optimization
  - Scheduling algorithm
  - Proxy placement and dimensioning

- Problems with complex scheduling mechanisms
- Examples
  - Caching with λ–patching
    - Patching window is optimized for minimal server load
  - Caching with gleaning
    - A 1st level proxy cache maintains the ”client buffer” for several clients
  - Caching with MPatch
    - The initial portion of the movie is cached in a 1st level proxy cache

# Distribution Architectures: λ–Patching

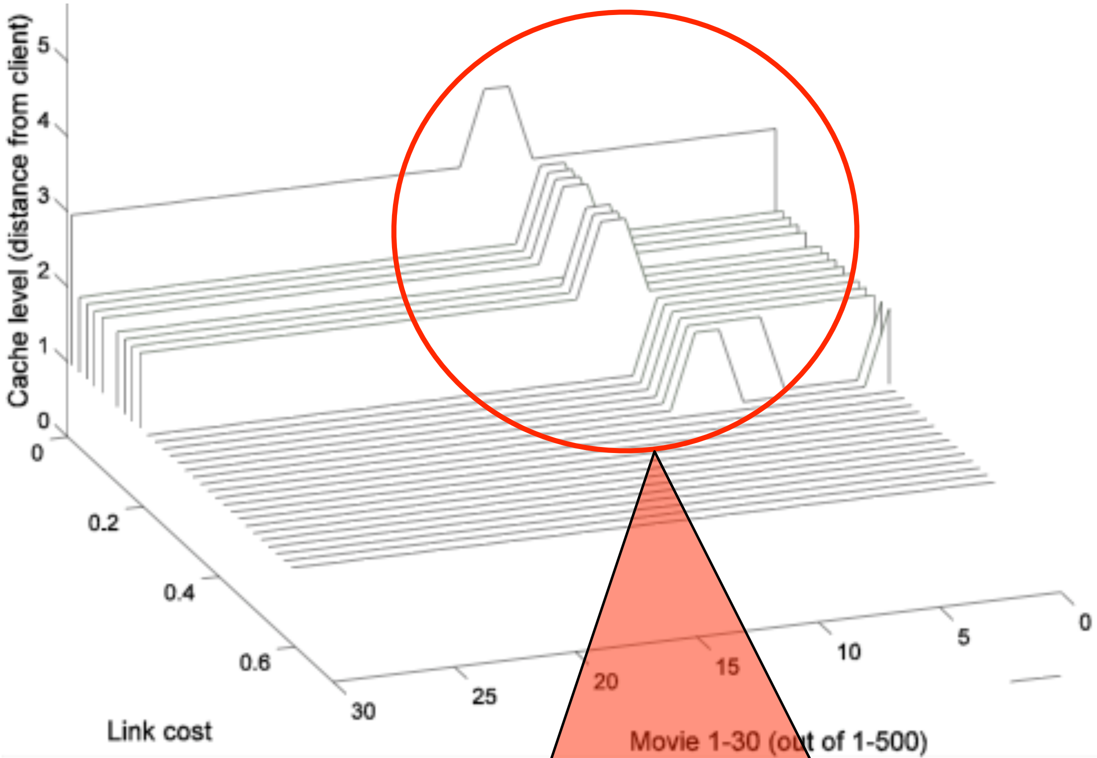



$$\Delta_M = \sqrt{2 \times F \times \Delta_U}$$

# Distribution Architectures: λ–Patching

- Placement for λ–patching




Popular movies may be more distant to the client

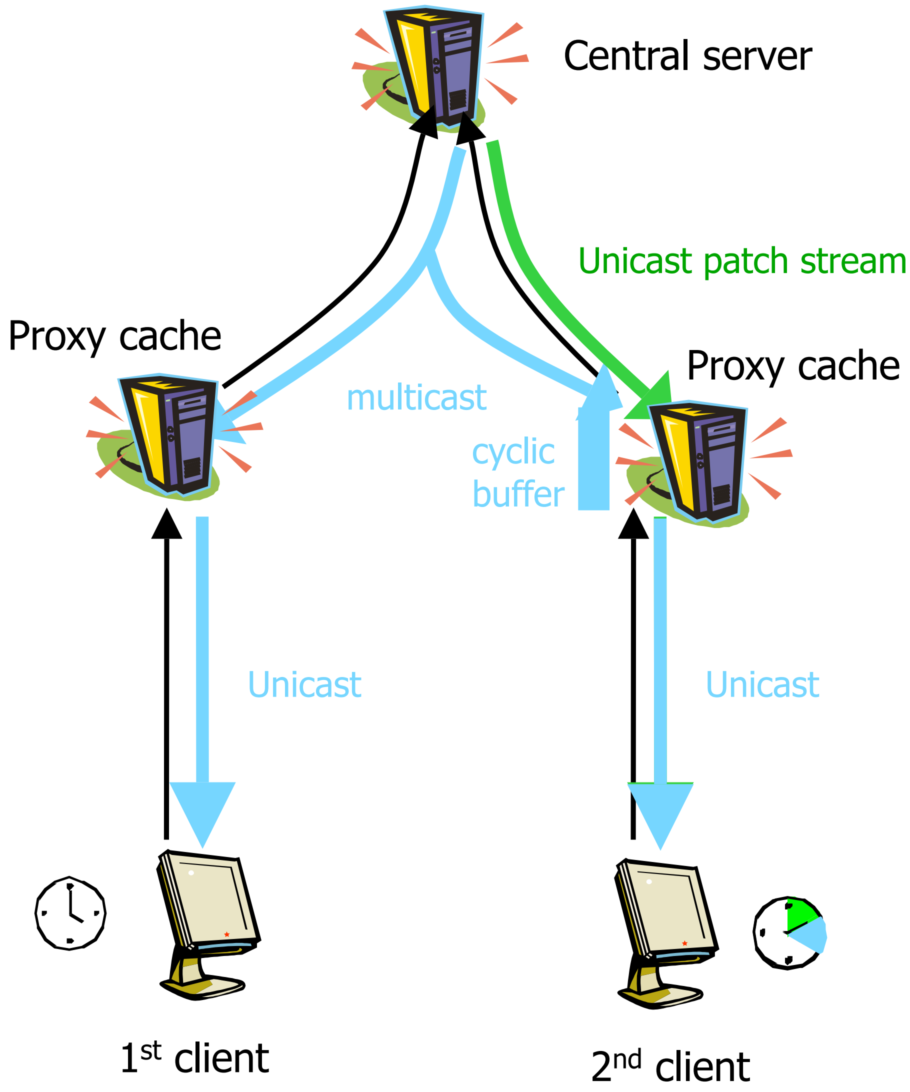
# Distribution Architectures: λ–Patching

- Failure of the optimization
  - Implicitly assumes perfect delivery
  - Has no notion of quality
  - User satisfaction is ignored

- Disadvantages
  - Popular movies further away from clients
    - Longer distance
    - Higher startup latency
    - Higher loss rate
    - More jitter
  - Popular movies are requested more frequently
  - **Average** delivery quality is lower

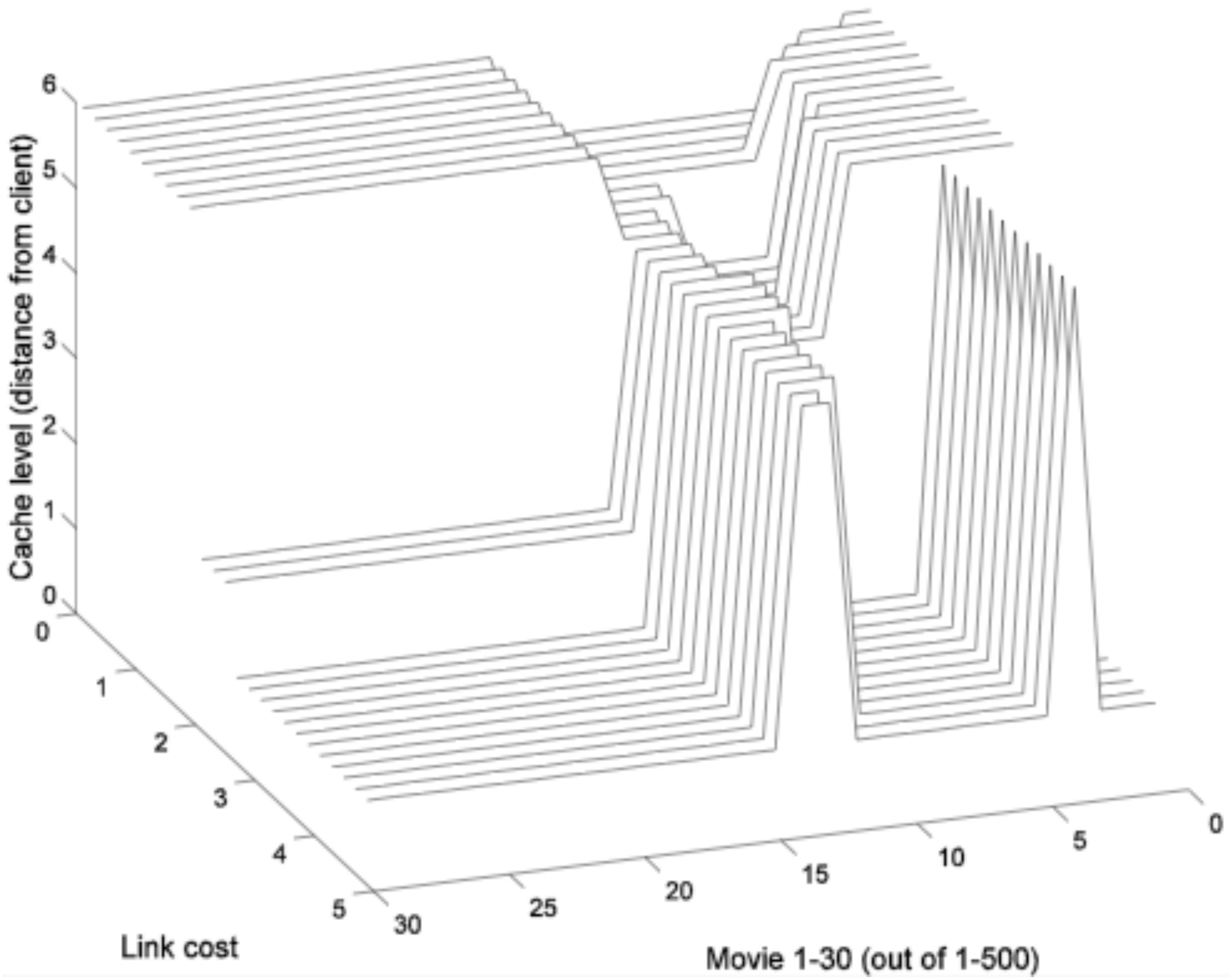
# Distribution Architectures: Gleaning

- Placement for gleaning
  - Combines
    - Caching of the full movie
    - Optimized patching
    - Mandatory proxy cache
  - 2 degrees of freedom
    - Caching level
    - Patch length



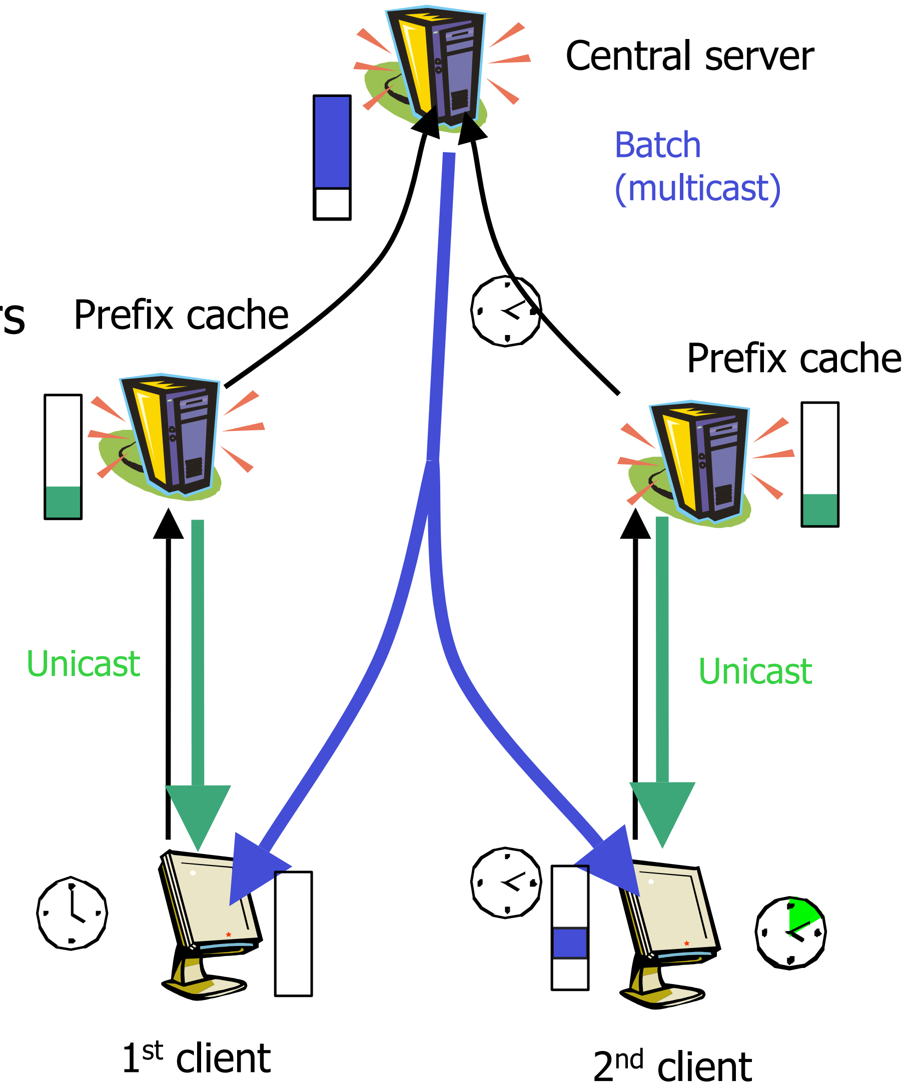
# Distribution Architectures: Gleaning

- Placement for gleaning



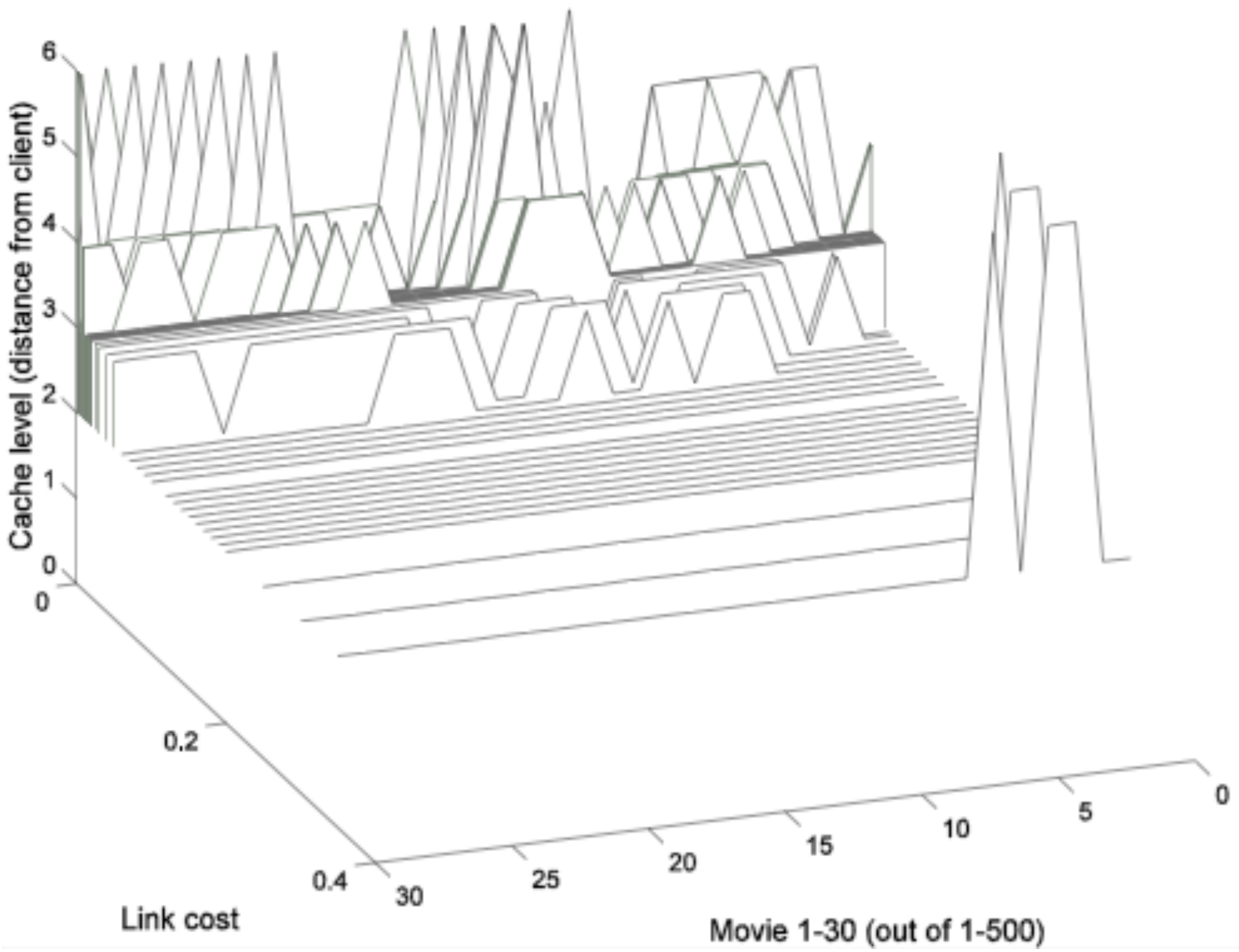
# Distribution Architectures: MPatch

- Placement for MPatch
  - Combines
    - Caching of the full movie
    - Partial caching in proxy servers
    - Multicast in access networks
    - Patching from the full copy
  - 3 degrees of freedom
    - Caching level
    - Patch length
    - Prefix length



# Distribution Architectures: MPatch

- Placement for MPatch



# Approaches

- Current approached does not consider quality
  - Penalize distance in optimality calculation
  - Sort
- Penalty approach
  - Low penalties ☹
    - Doesn't achieve order because actual cost is higher
  - High penalties
    - Doesn't achieve order because optimizer gets confused
- Sorting
  - Trivial ☺
  - Very low resource waste

# Distribution Architectures

- Combined optimization
  - Scheduling algorithm
  - Proxy placement and dimensioning
  - Impossible to achieve optimum with autonomous caching

- Solution for complex scheduling mechanisms

- A simple solution exists:
  - Enforce order according to priorities
    - (simple sorting)
  - Increase in resource use is marginal

# Summary

- A lot of performance can be gained using appropriate distribution mechanisms
  - type 1: delayed delivery
  - type 2: prescheduled delivery
  - type 3: client side caching
  - type 4: network of servers combining types 1-3

- Caching is (almost) always beneficial
  - better with conditional replacements / overwrites
  - hints from neighboring caches