*Towards A Common API for Structured Peer-to-Peer Overlays

Frank Dabek, Ben Y. Zhao,
Peter Druschel, John Kubiatowicz,
Ion Stoica

MIT, U. C. Berkeley, and Rice

**** Conducted under the IRIS project (NSF)***

State of the Art

- Lots and lots of peer to peer applications
  - Decentralized file systems, archival backup
  - Group communication / coordination
  - Routing layers for anonymity, attack resilience
  - Scalable content distribution
- Built on scalable, self-organizing overlays
  - E.g. CAN, Chord, Pastry, Tapestry, Kademlia, etc…
- Semantic differences
  - Store/get data, locate objects, multicast / anycast
  - How do these functional layers relate?
  - What is the smallest common denominator?

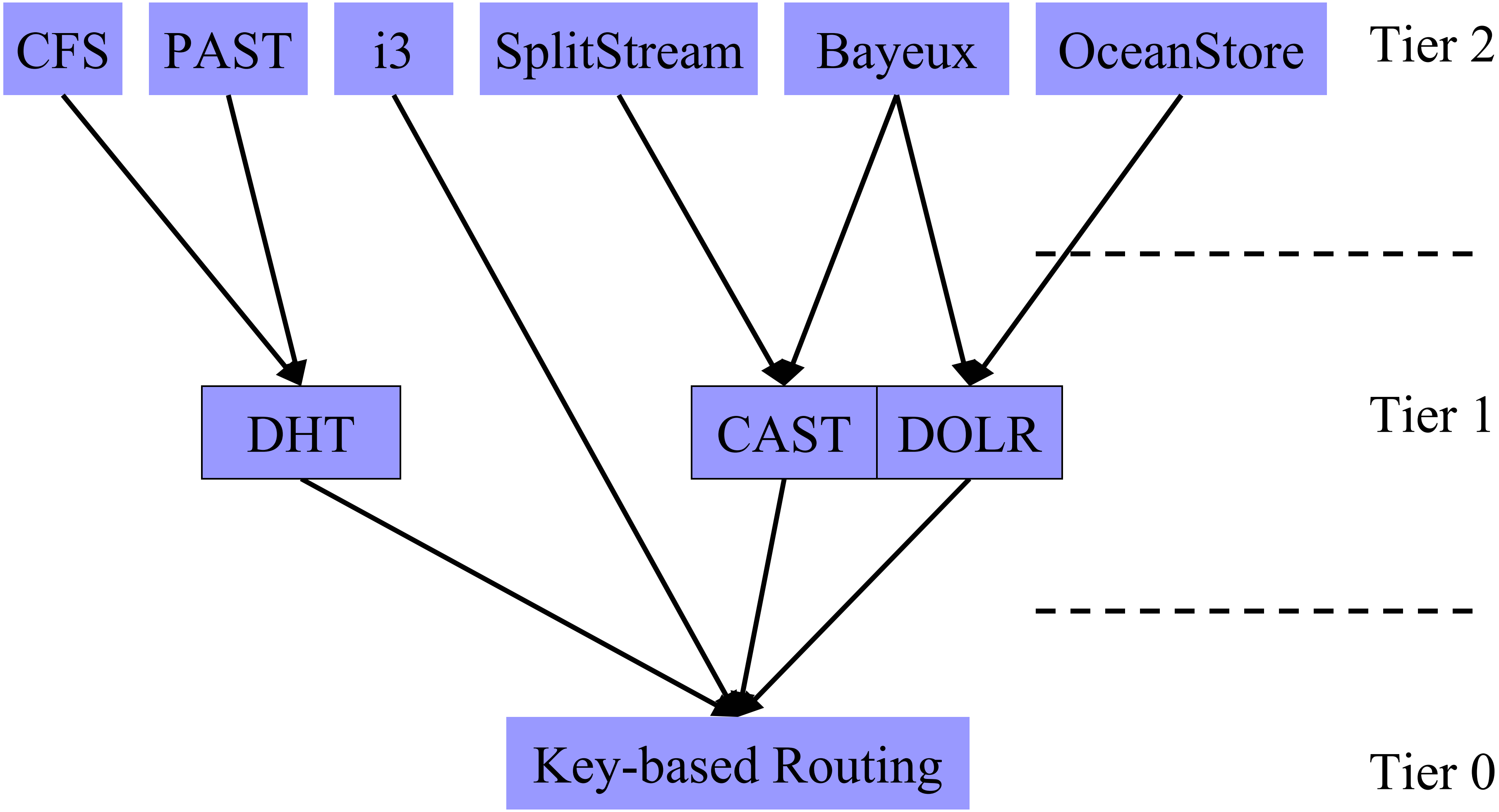
Some Abstractions

- Distributed Hash Tables (DHT)
  - Simple store and retrieve of values with a key
  - Values can be of any type
- Decentralized Object Location and Routing (DOLR)
  - Decentralized directory service for endpoints/objects
  - Route messages to *nearest* available endpoint
- Multicast / Anycast (CAST)
  - Scalable group communication
  - Decentralized membership management

Tier 1 Interfaces

Distributed Hash Tables (DHT)	Decentralized Object Location / Routing (DOLR)	Multicast / Anycast (CAST)
put (key, data)	*publish (objectId)*	*join (groupId)*
remove (key)	*unpublish (objectId)*	*leave (groupId)*
value = get (key)	*sendToObj (msg, objectId, [n])*	*multicast (msg, gId)* *anycast (msg, gId)*

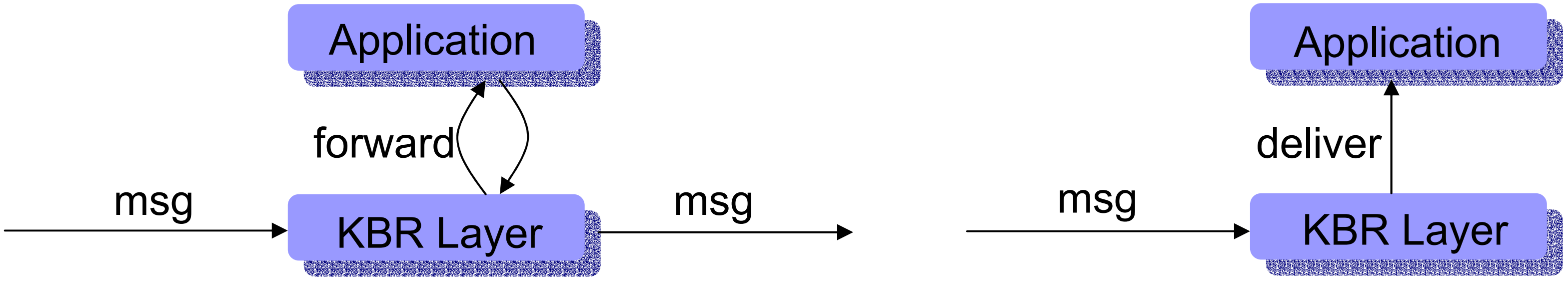
Structured P2P Overlays



The Common Denominator

- Key-based Routing layer (Tier 0)
  - □ Large sparse Id space *N* (160 bits: 0 – 2^{160} represented as base *b*)
  - □ Nodes in overlay network have nodeIds $\in N$
  - □ Given $k \in N$, overlay deterministically maps *k* to its *root* node (a live node in the network)
- Goal: Standardize API at this layer
- Main routing call
  - □ **route (key, msg, [node])**
  - □ Route message to node currently responsible for key
- Supplementary calls
  - □ Flexible upcall interface for customized routing
  - □ Accessing and managing the ID- space

Flexible Routing via Upcalls

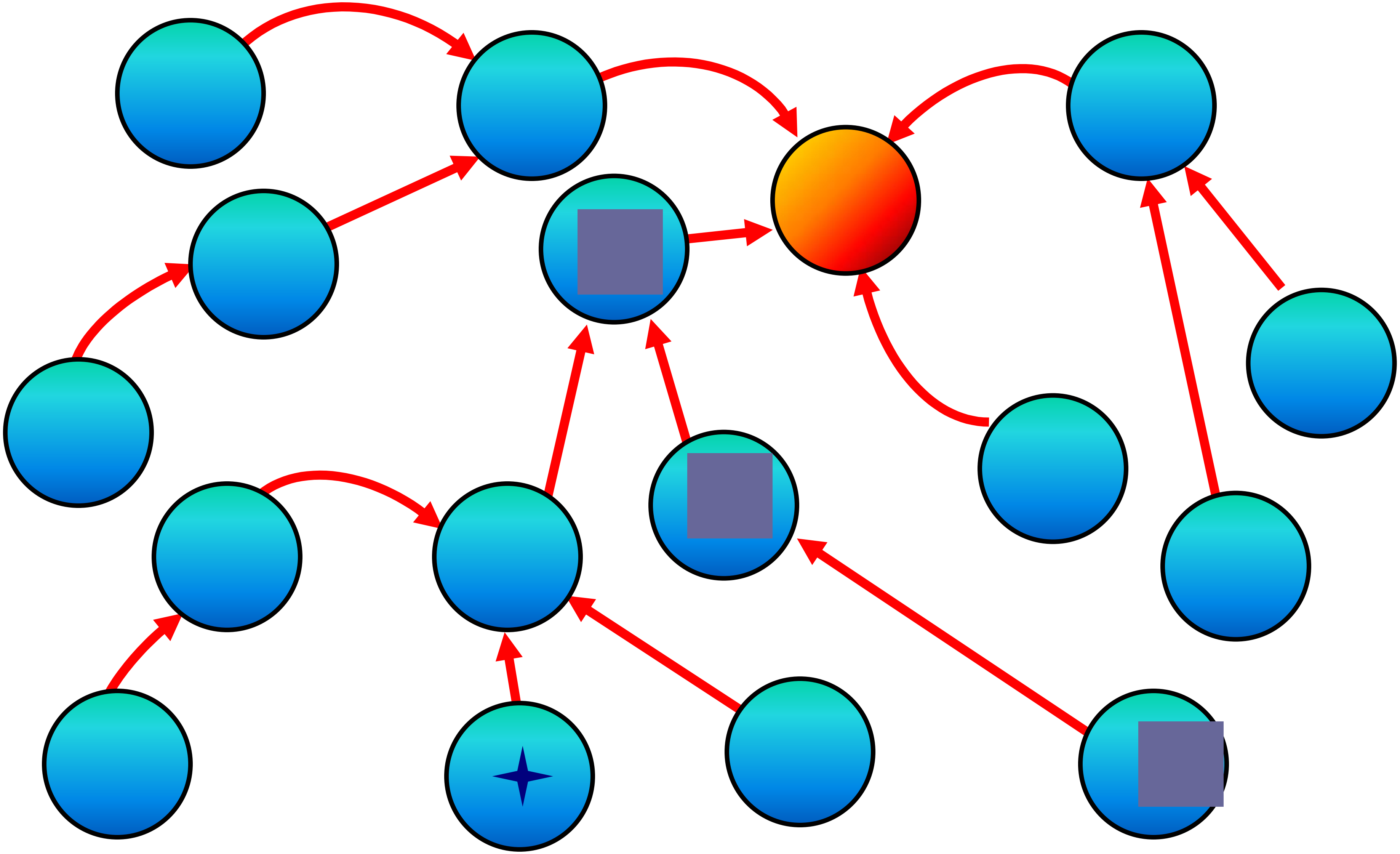


- **Deliver(key, msg)**
  - Delivers a message to application at the destination
- **Forward(&key, &msg, &nextHopNode)**
  - Synchronous upcall with normal next hop node
  - Applications can override messages
- **Update(node, *boolean* joined)**
  - Upcall invoked to inform application of a node joining or leaving the local node’s neighborSet

KBR API (managing ID space)

- Expose local routing table
  - **nextHopSet = local_lookup (key, num, safe)**
- Query the ID space
  - **nodehandle[ ] = neighborSet (max_rank)**
  - **nodehandle[ ] = replicaSet (key, num)**
  - **boolean = range (node, rank, lkey, rkey)**

Caching DHT Illustrated



Caching DHT Implementation

- Interface
  - □ *put (key, value)*
  - □ *value = get (key)*
- Implementation (source S, client C, root R)
  - □ Put: *route(key, [PUT,value,S])*
    Forward upcall: store value
    Deliver upcall: store value
  - □ Get: *route(key, [GET,C])*
    Forward upcall: if cached, *route(C, [value]), FIN*
    Deliver upcall: if found, *route(C, [value])*

Ongoing Work

- What's next
  - Better understanding of DOLR vs. CAST
    - Decentralized endpoint management
    - Policies in placement of indirection points
  - APIs and semantics for Tier 1: (DHT/DOLR/CAST)
  - KBR API implementation in current protocols
- See paper for additional details
  - Implementation of Tier 1 interfaces on KBR
  - KBR API support on selected P2P systems