goto; amsterdam

Please

Ask questions through the app

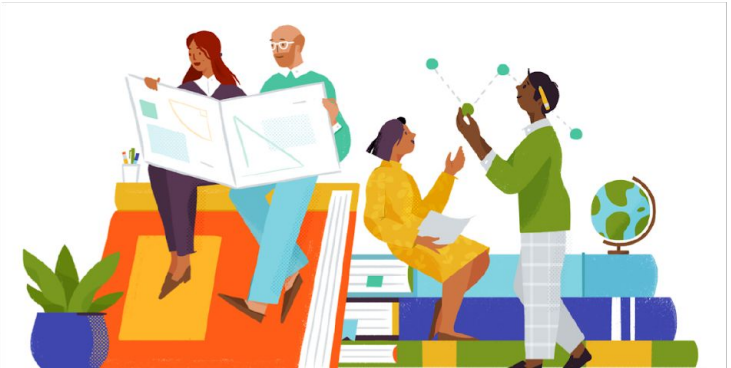
Rate Session

Thank you!

Scaling Slack

Keith Adams -- kma@slack-corp.com

GOTO Amsterdam 2018



Introduction

Some impossibility results

How Slack works even though it can’t

Two case studies

Takeaways

Acme Sites
Victoria Thomas

All Threads

STARRED
design-work
summer-campaign
3 Cory, Tina, Carl

CHANNELS
accounting-costs
brainstorming 1
business-ops
client-proposal
design-chat
marketing
media-and-pr
sonic-fanfic
triage-issues

DIRECT MESSAGES
slackbot

#client-proposal

Victoria Thomas
Hey team, hoping to have that proposal ready for the Alaska clients by 3pm today, how are we doing? I can chip in wherever needed!

Carl Benting
I'm just about finished putting together the estimate portion of it, I could use some feedback. Here's the google doc I'm working on... docs.google.com/bin/proposal

Q3 OOH — Cost Estimate
Google Drive Document

✅ 1

Victoria Thomas
The numbers look pretty good, I tweaked a few things, but we're good to go!

Reena Baines
I'm just wrapping up the sketches, I'll post them here once I'm done!

About #client-proposal

- Channel Details
- Pinned Items
- 12/19 Members
- Shared Files
- Notification Preferences

Like IRC?

Only visually.

- IRC is defined by its *ephemerality*
- Slack offers *persistence*
- Like a hybrid of e-mail and IRC

Slack Technical Constraints

Minimal Behavior of a Channel

- Validity/Agreement: if a member sends/receives a message, all members will eventually receive it.
- Integrity: a message is received by each member at most once, and only if it was previously sent
- Total Order: all members receive messages in the same order

Atomic Broadcast Definition

- Validity/Agreement: if a member sends/receives a message, all members will eventually receive it.
- Integrity: a message is received by each member at most once, and only if it was previously sent
- Total Order: all members receive messages in the same order

Uh-oh.

- Atomic broadcast is equivalent to consensus[1]
- Consensus in general is impossible[2]

[1] Chandra and Toueg. Unreliable failure detectors for reliable distributed systems. JACM 43(2):225–267, 1996.

[2] Fischer, Lynch, and Paterson. Impossibility of Distributed Consensus with One Faulty Process. JACM 32(2):374-382, 1985.

So … are we done here?

RIP Slack
2014-2018
“Useful until proven impossible”

Of course not!

- There are *practically useful* consensus systems despite FLP
- 🔑: Relax constraints
- Cryptocurrencies: probabilistic log
- Paxos/ZAB/Raft/...: might not terminate

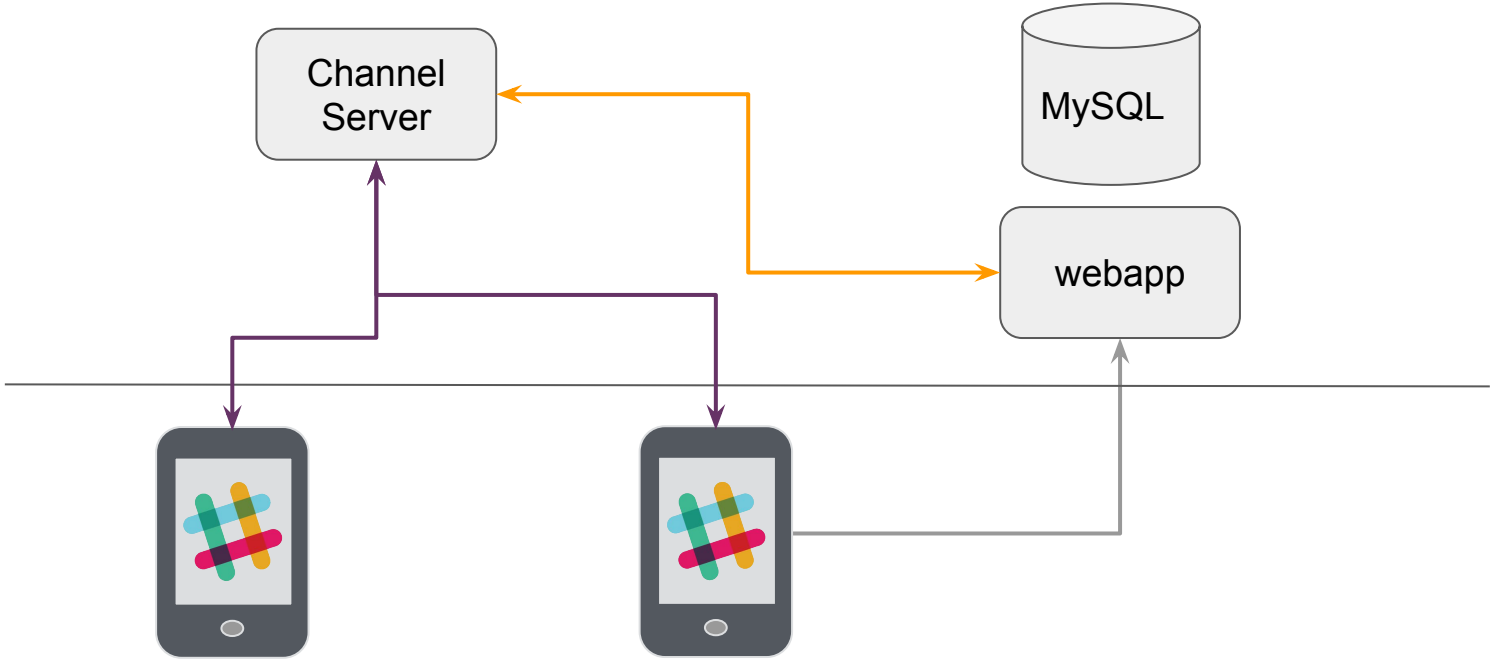
Scaling Impossible Things

- What constraints to relax is an *end-to-end property*[1] of the system
- Varies by application, its parameters
- Complexity is inherent
- Our solution keeps changing with app, scale, user behavior, hardware economics, ...

J. H. Saltzer, D. P. Reed, D. D. Clark, "End-to-End Arguments in System Design," 2nd International Conference on Distributed Computing Systems, Paris, (April 1981), pp. 509-512.

Case Study #1: Message Send/Receive

Slack Cartoon



Division of Labor

WebApp

>1MLoC Hacklang monolith. Medium levels of SOA-osity.

- CRUD
- Storage
- Retrieval, permissions
- Session establishment

Channel Server

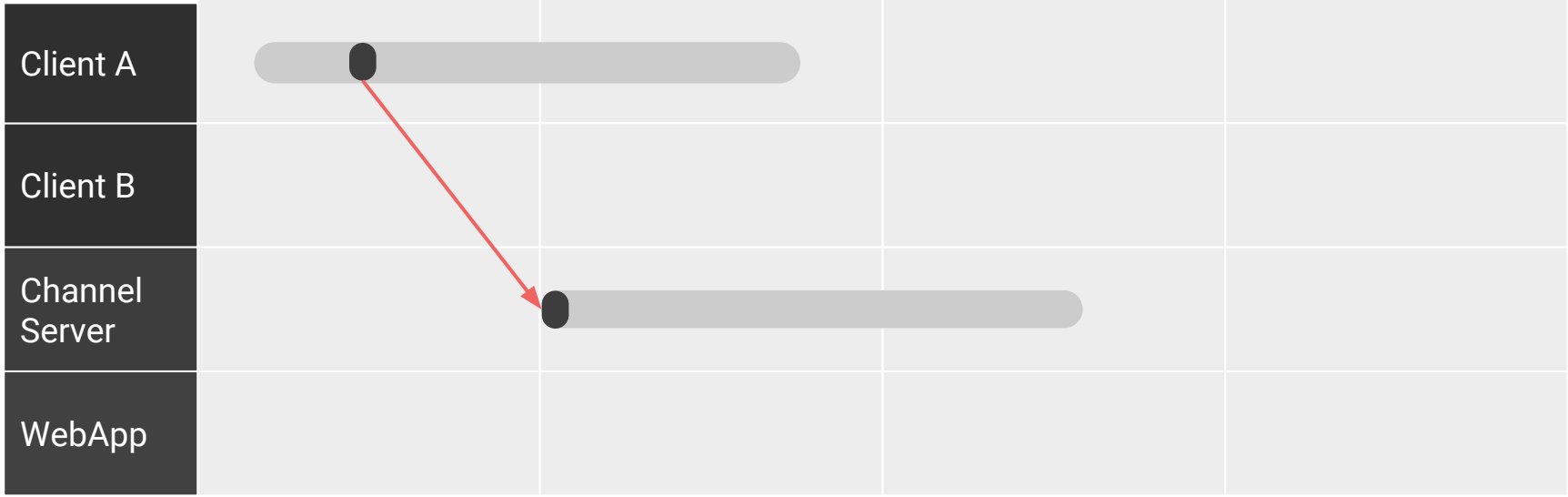
Real-time service, accessed over WebSockets.

- Push updates to clients
- Messages, typing indicators, presence
- Witness to *order of messages*
- Grab-bag of other roles

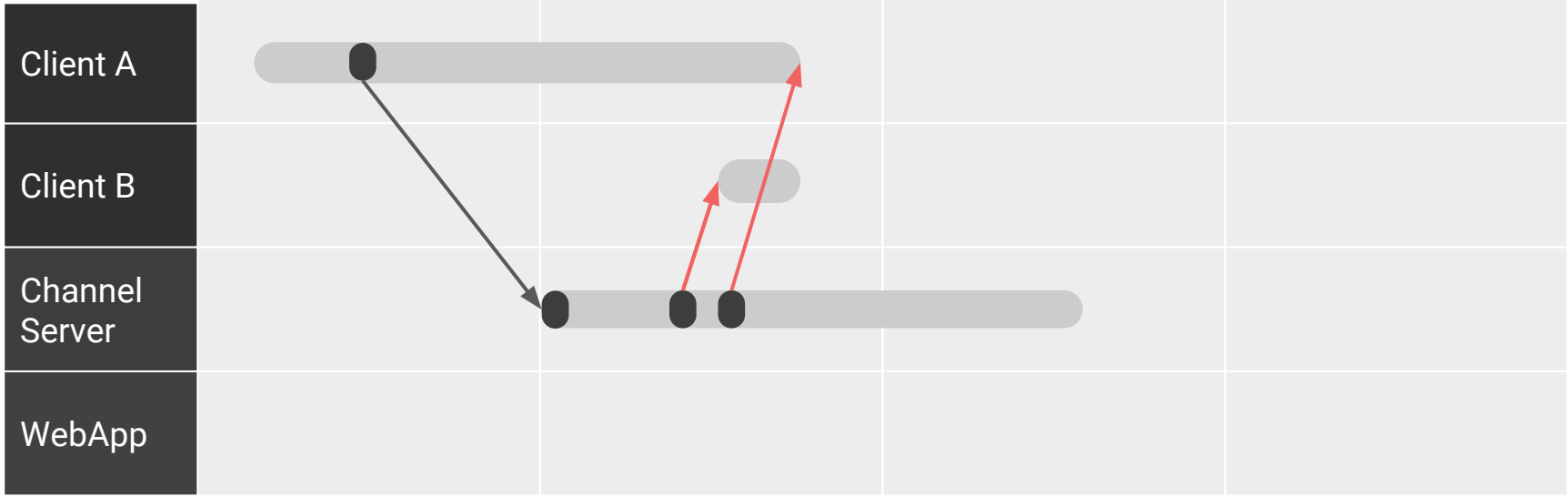
Send/receive for Online Clients

Client A				
Client B				
Channel Server				
WebApp				

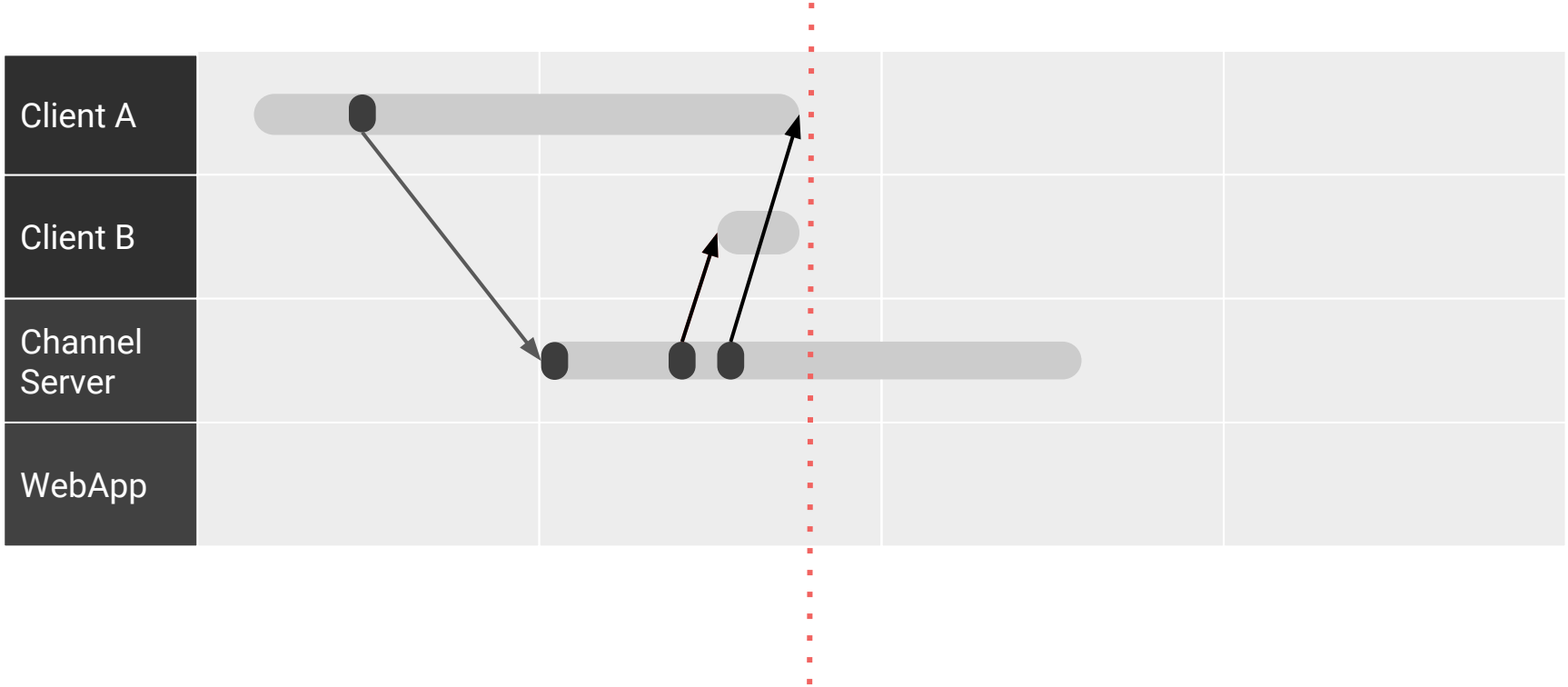
Client Sends to CS



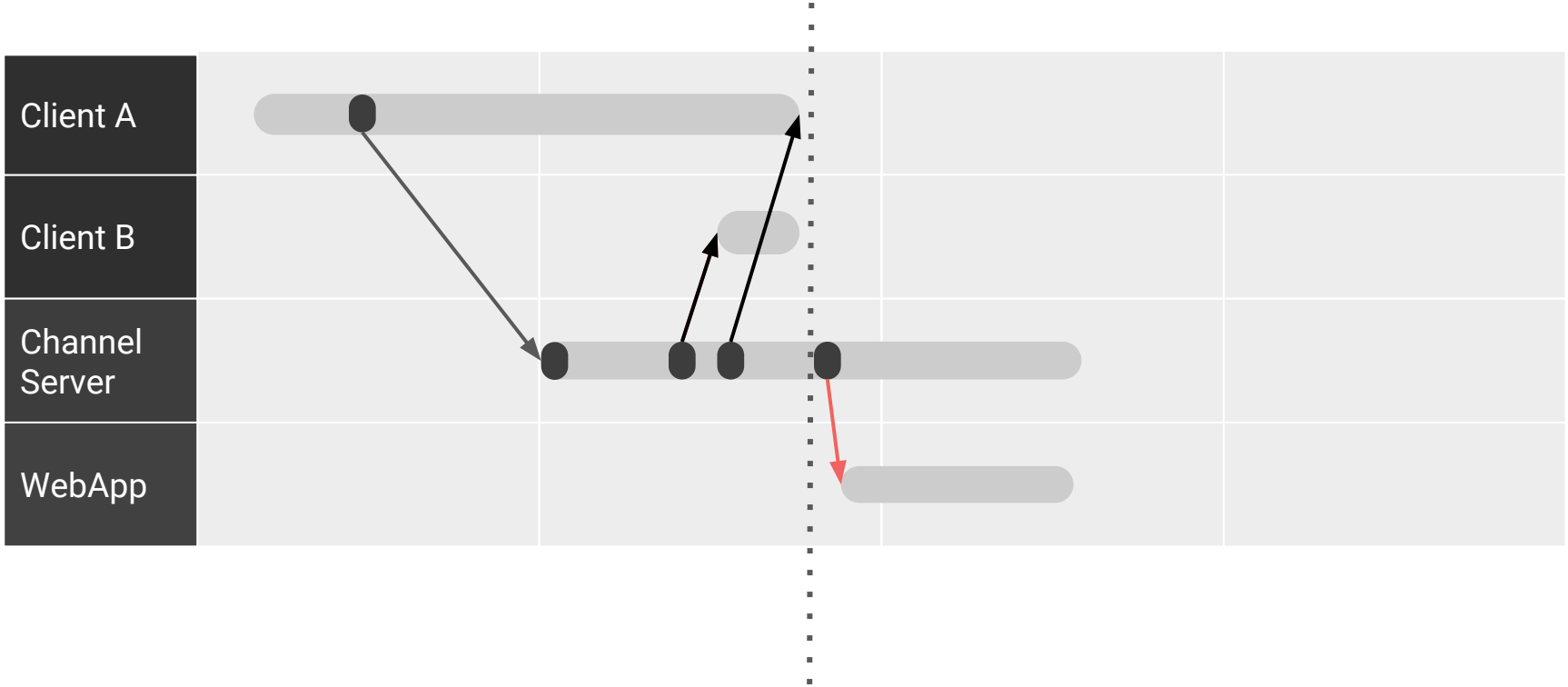
CS Amplifies, then Acks



End of User-Perceived Latency



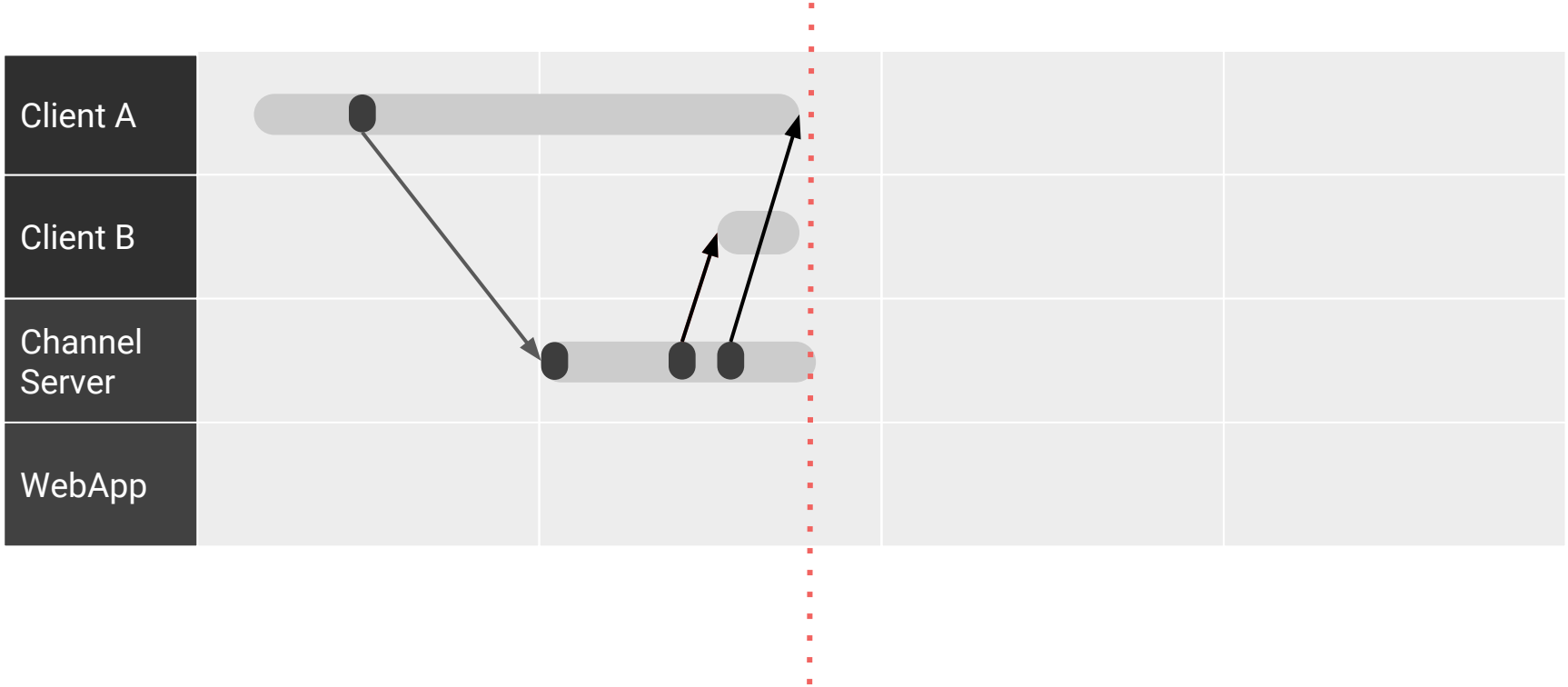
Store Message in DB



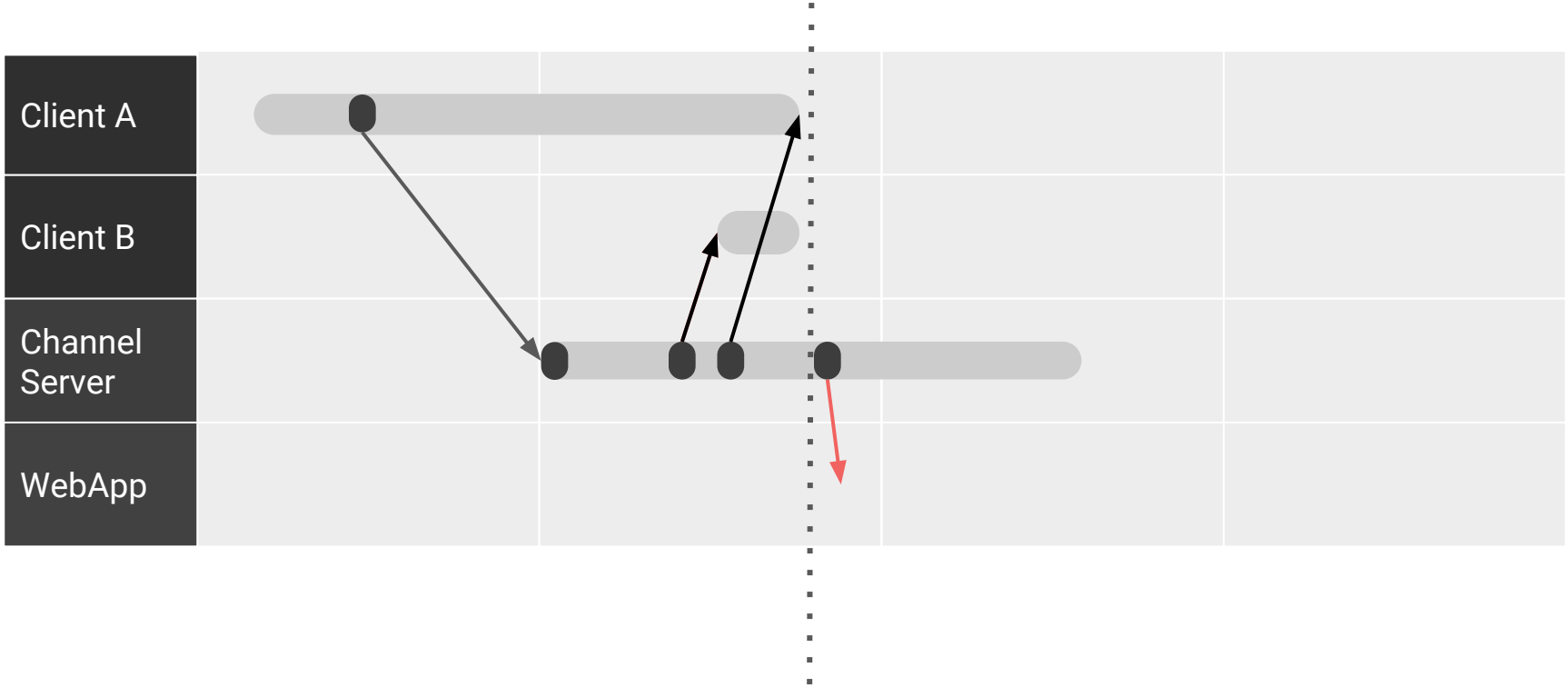
The Happy Path

- Latency of WebApp, DB writes hidden from users
- But what if something goes wrong?

CS Crash!



WebApp Outage



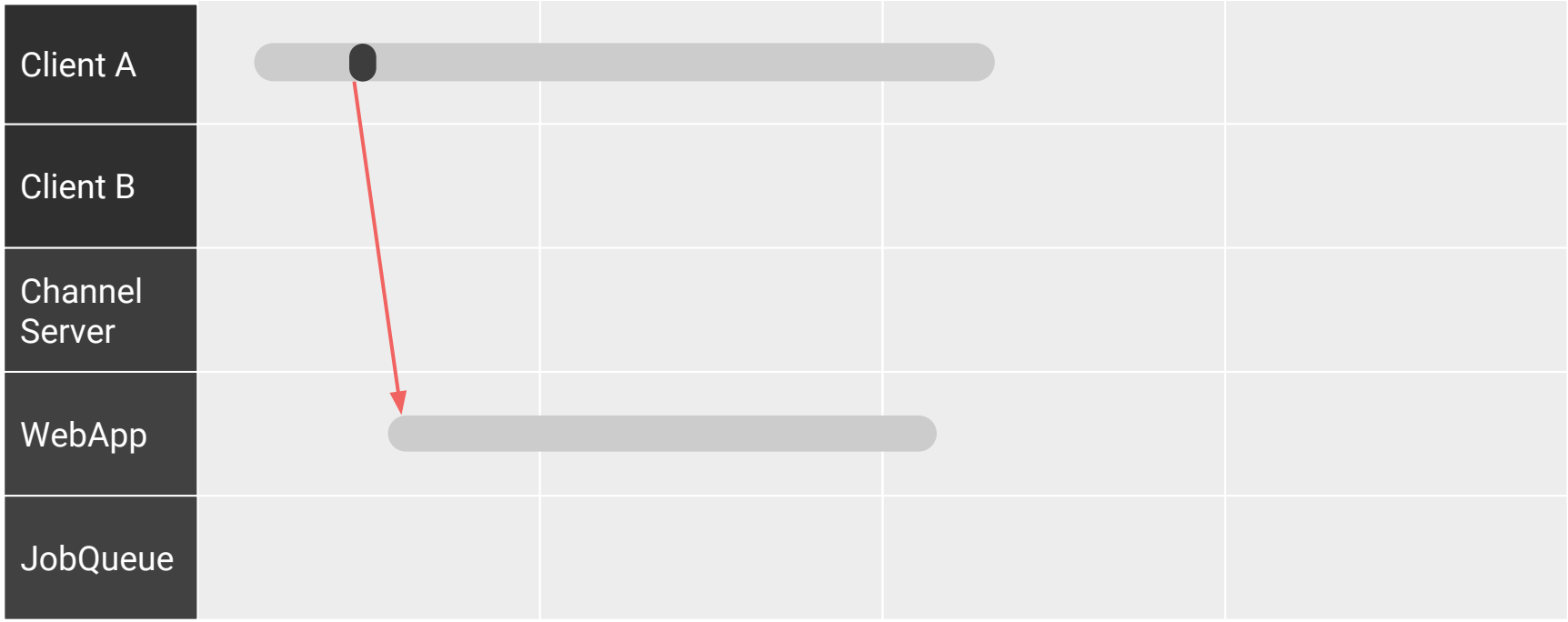
Dealing with Failures

- CS maintains an on-disk buffer of uncommitted sends
- Replayed when recovering from CS crash
- Retried while webapp is unavailable
- State
  - Complexity
  - Risk during CS code changes
  - But provides partial end-to-end utility while site is hard-down

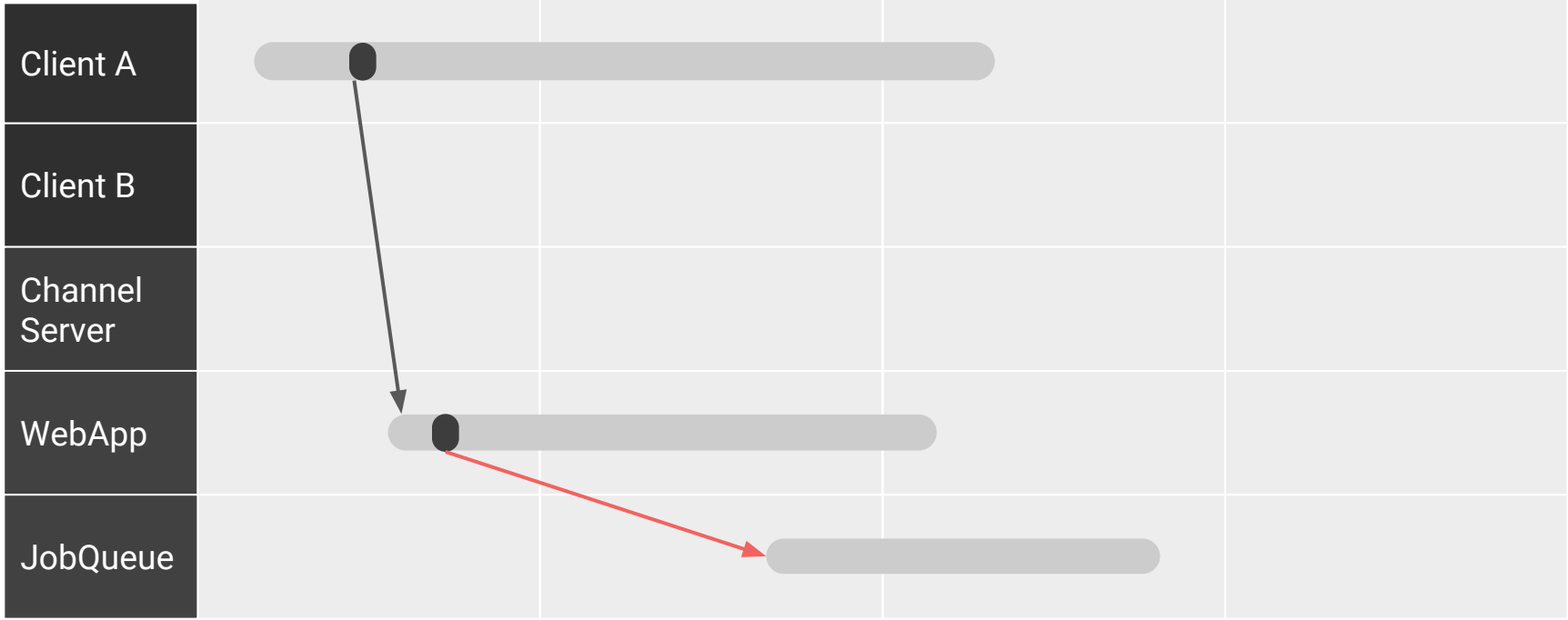
Changes since 2014

- Webapp more stable
- Job queue more stable and scalable
  - Safe way of deferring work
  - See Saroj, Matt, Mike, and Tyler's blog post

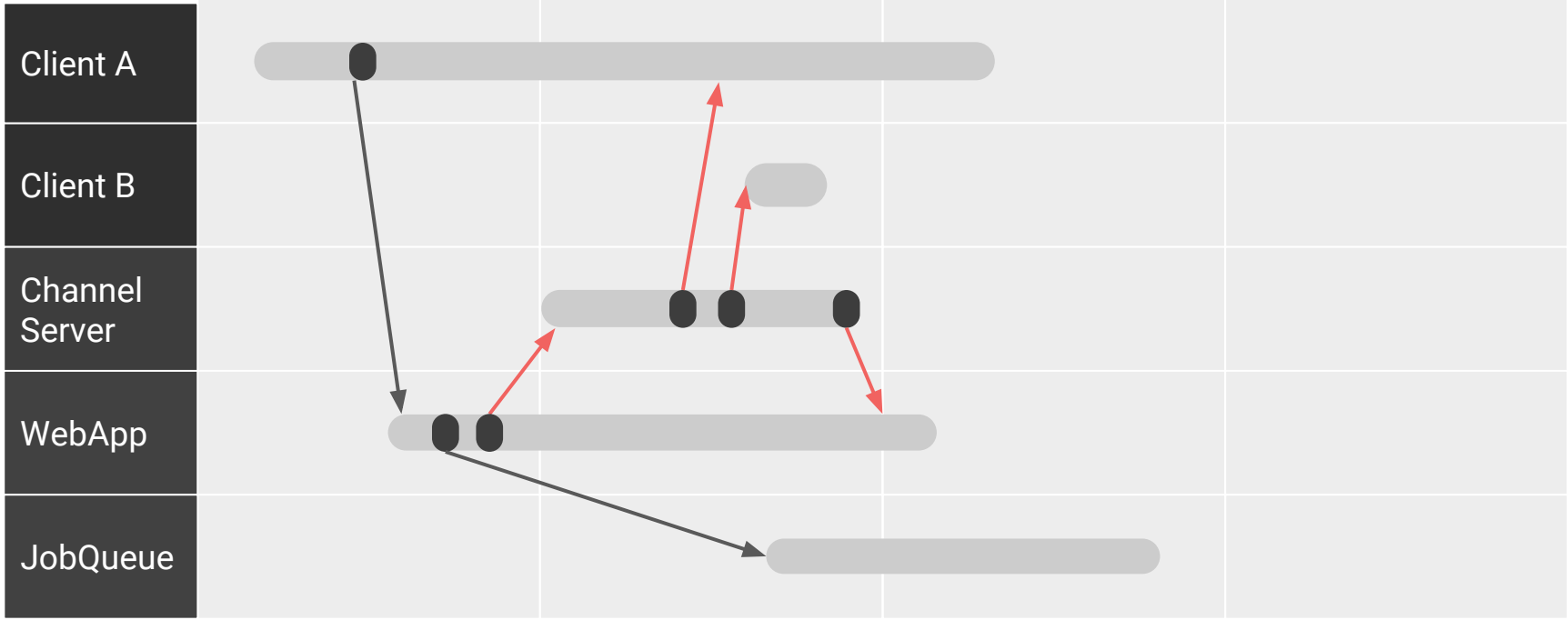
New Send Flow



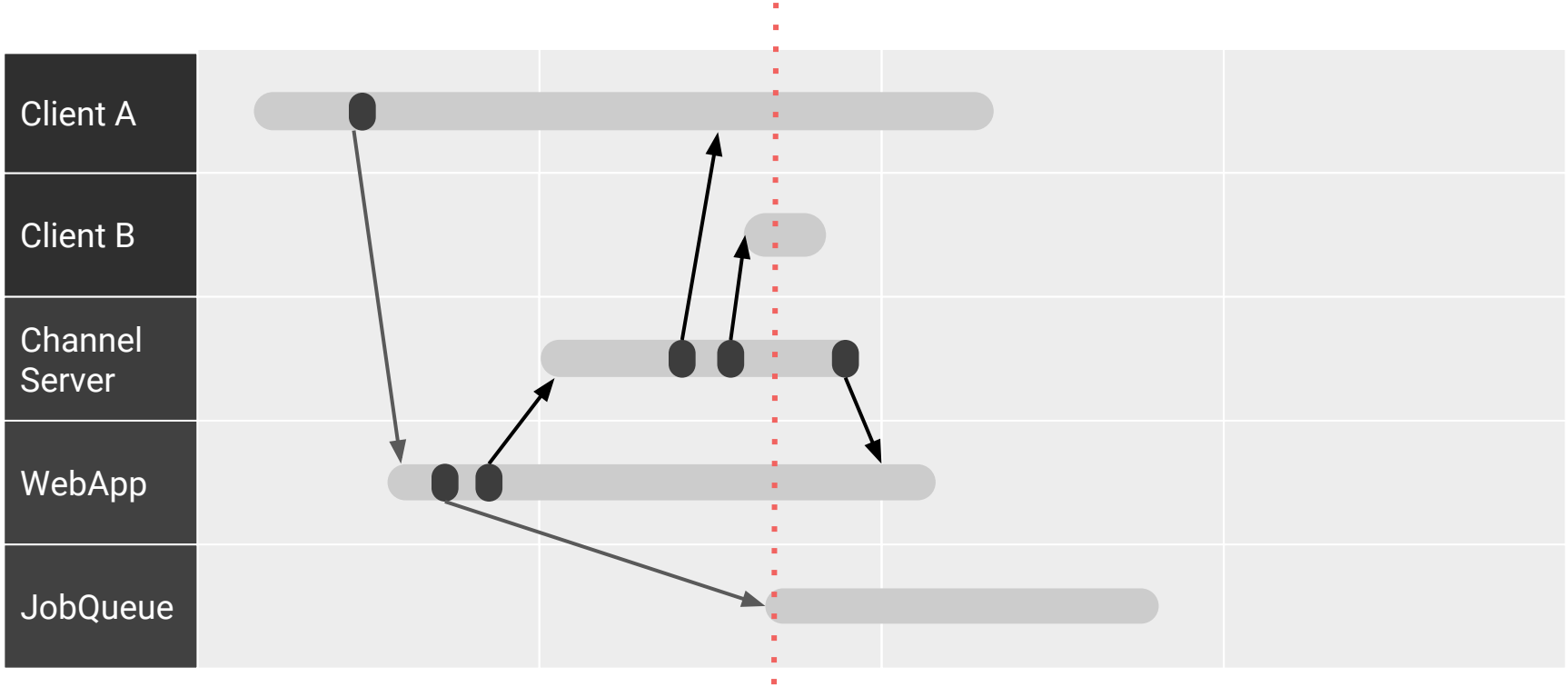
Defer Slow Work



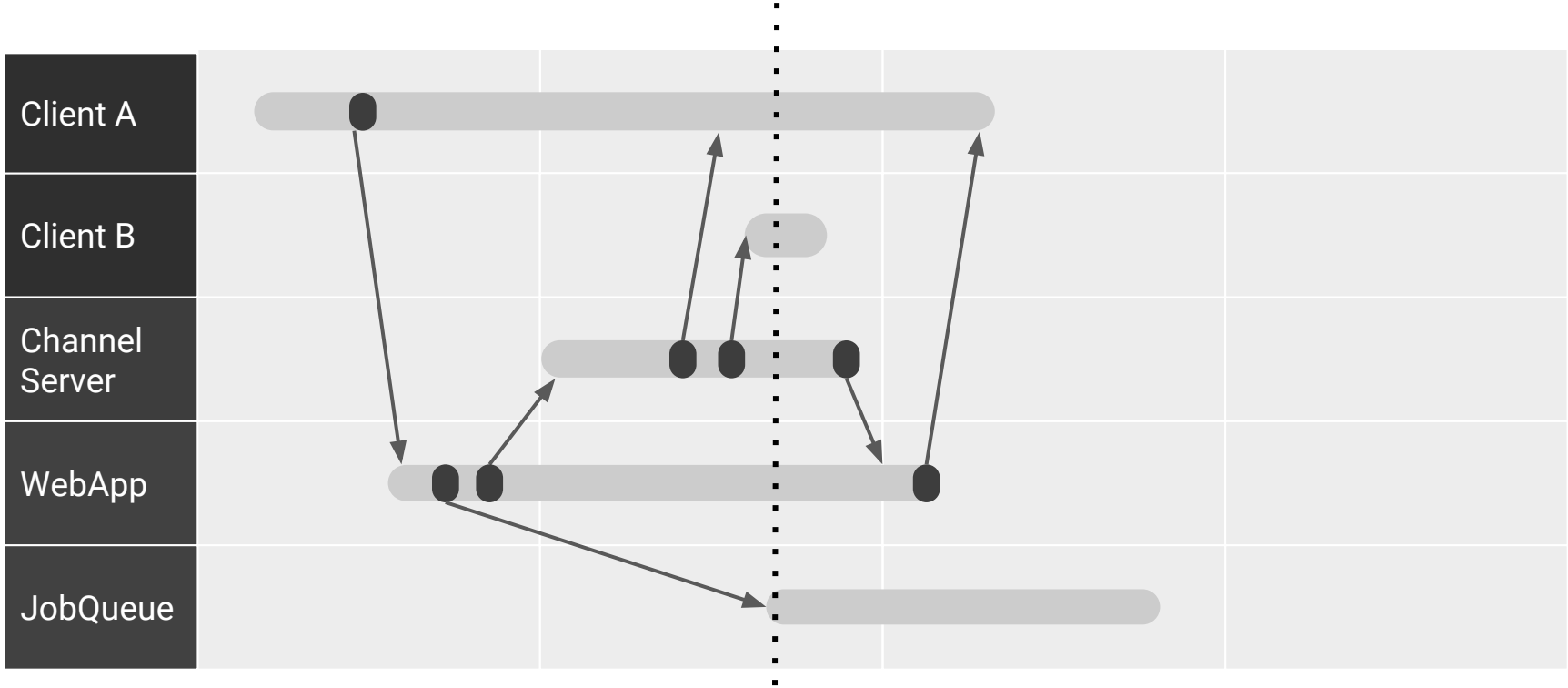
Send Real-Time Updates



End of user-perceived latency



HTTP 200, deferred work



New Flow Observations

- Crash-safe
- Low latency by deferring costly parts
- Stateless-ish CS now possible
- Clients can send without establishing a web socket session

So this way is *better*, right?

- In 2018, yes
  - Still rolling out to all geographies, teams
- But it definitely wasn’t in 2014
  - Extra hop between clients
  - Webapp was less available
  - JobQueue was finite capacity

Case Study #2: WebSocket initiation

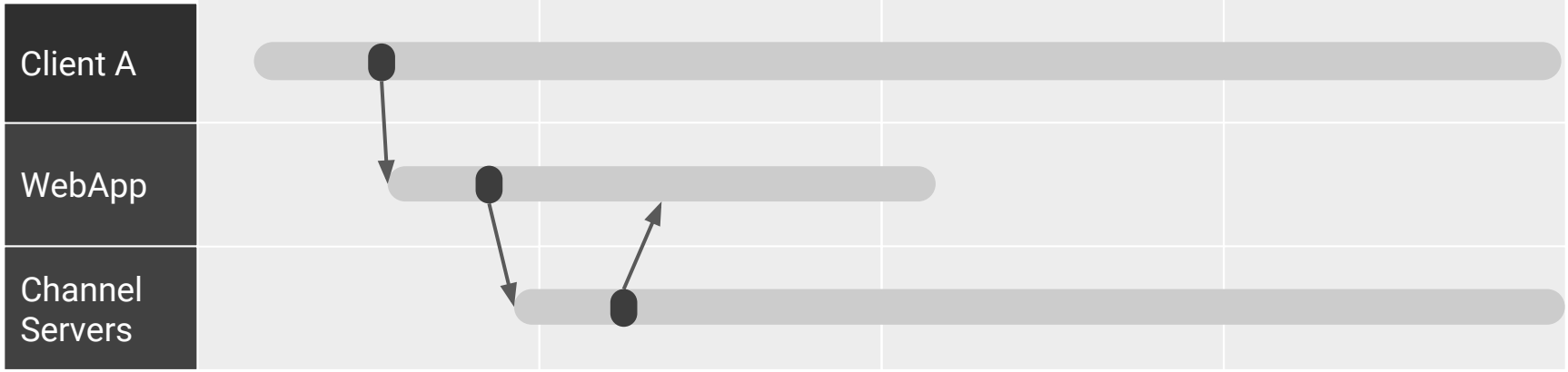
Slack is Connection-Oriented

- Most of our community's scaling experience is *request*-oriented
- Slack: Server-push via WebSockets
- > 5M simultaneous sessions at peak, with wide peak-to-trough variations

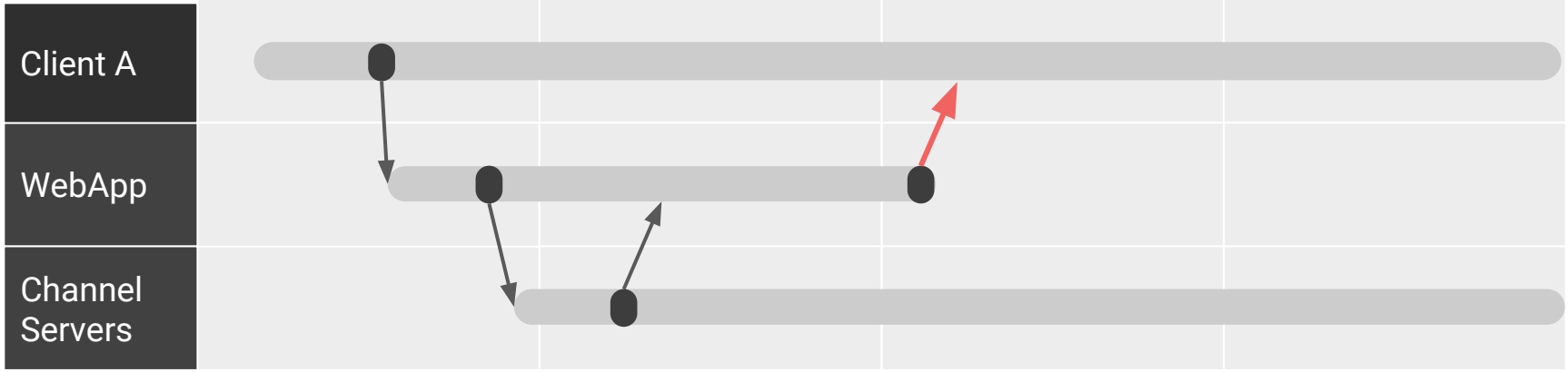
Classic Session Establishment Pattern

- Invoke rtm.start API method
- Use wss:// url in results to start session

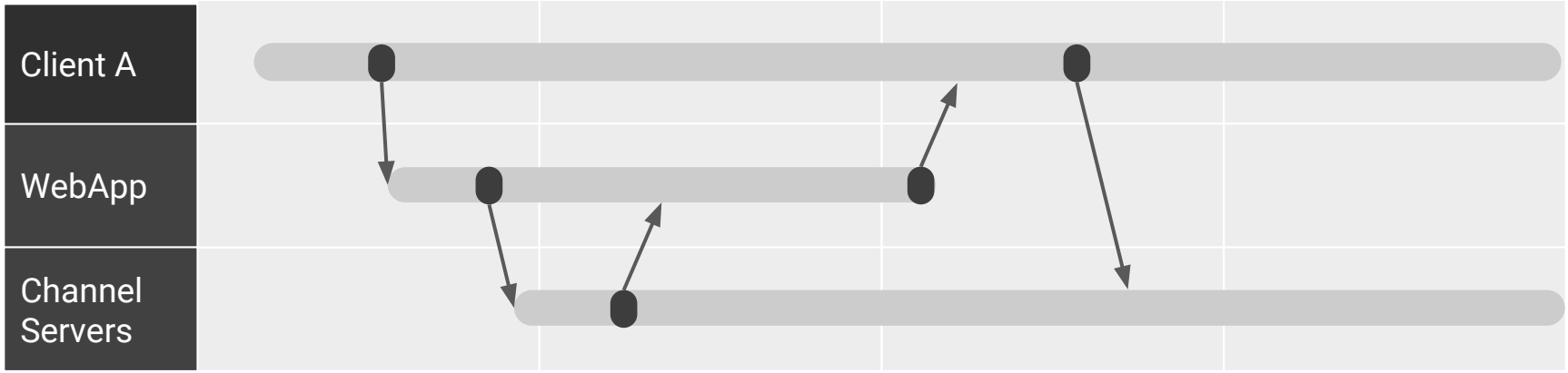
WebApp harvests team data



WebApp delivers *huge* payload



Establishing WS connection (done)



Rtm.start payload

- “Keyframe” of team state
- Users, profiles, channels and membership, latest-modified timestamps for channels, logged-in users’ last-read timestamps, ...
- Incremental updates via WebSocket

Great in 2014!

- *This worked great for small teams*
- …close to Slack’s datacenter
- As organizations surpassed 1000, then 10,000, then 100,000
- ...and spread across the globe...

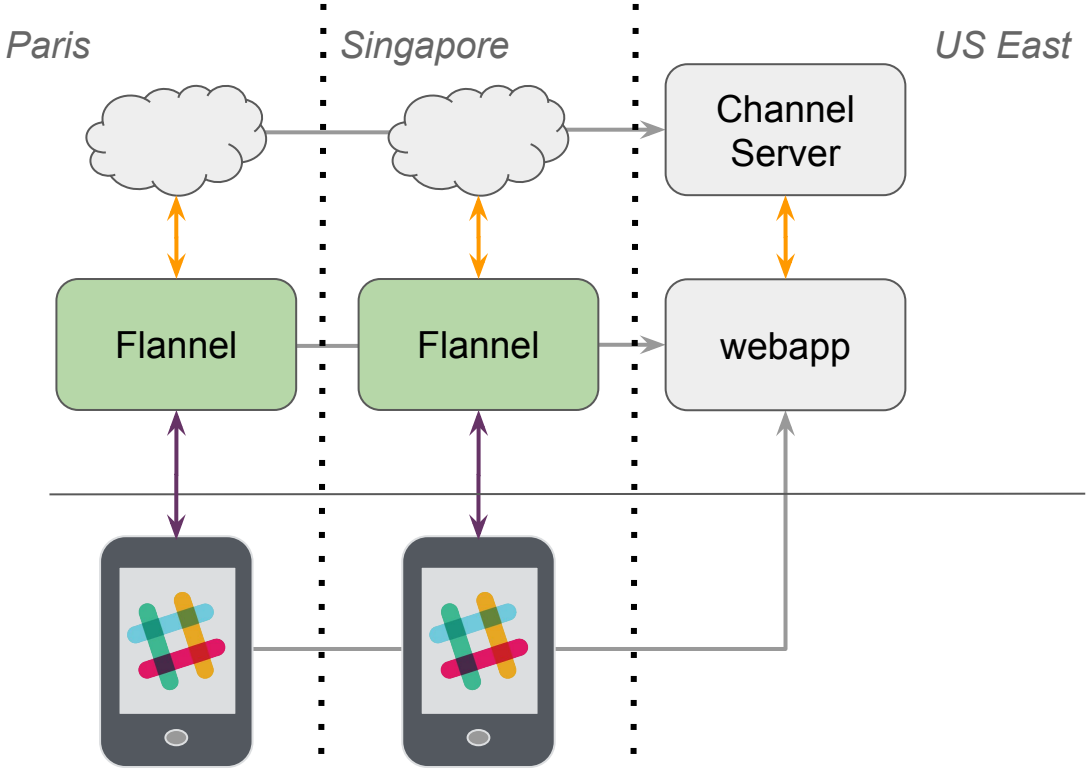
Problems

1. **Rtm.start payload size.** *(Performance)*
2. **Connection storms place redundant load on databases.** *(Reliability)*
3. **Round-trip times for most of the world.** *(Performance)*

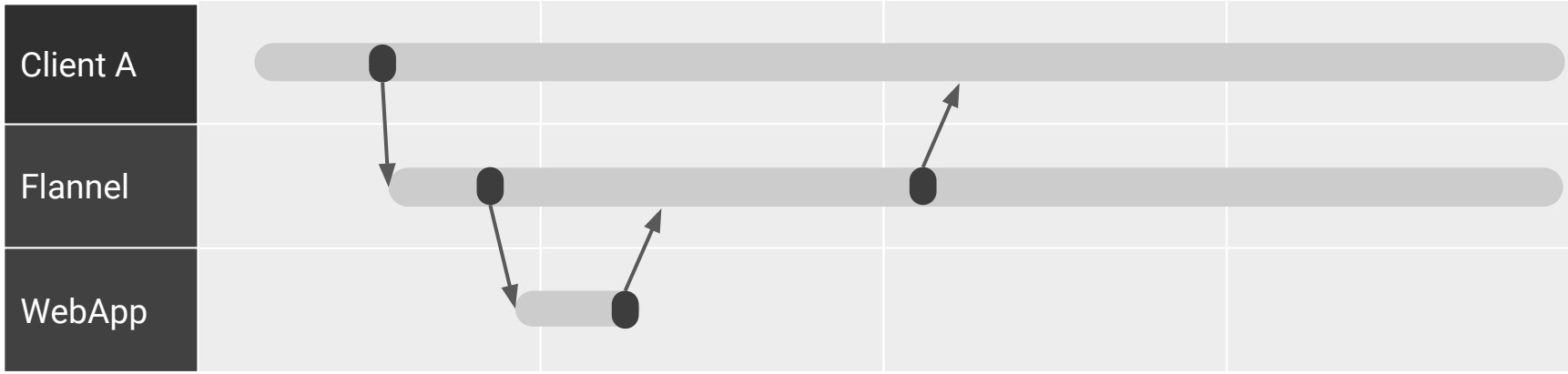
Slack’s Solution: Flannel

- Stateful, Application-aware Microservice
- Pre-warmed cache of teams, channels, users, ...
- Terminates websockets
- Runs in edge regions, reducing load on core and improving service time
- See Bing Wei’s blog post and talk for more details

Flannel


Paris
Singapore
US East
Channel Server
Flannel
Flannel
webapp

Establishing Session Flannel-Style



So Flannel is *better*, right?

- Yes!
- Simpler, safer, faster
- But no way to foresee this before reaching this scale
- Next scale might change

Takeaways

- Find the *end-to-end* part of your problem
- Optimality is contingent, and changes with growth
- Simplicity misapplied is just as poisonous as complexity

goto; amsterdam

Please

Remember to rate this session

Thank you!