gRPConf NORTH AMERICA

Streaming Was Only Half the Fix

Retrofitting a Stable gRPC API

Sohan Kunkerkar, Red Hat

A little about me

gRPConf NORTH AMERICA

ROLE

Senior Software Engineer

Red Hat





PROJECT MAINTAINER

CRI-O

The container runtime interface for Kubernetes

Kueue

Kubernetes-native job queueing controller

FOCUS AREAS

Kubernetes Node

Working on the kubelet, CRI, and the container runtimes underneath.

AI/ML Scheduling

Working on Kueue and Dynamic Resource Allocation (how GPUs get allocated to pods)

What we'll cover

gRPConf NORTH AMERICA

01
kubelet & Runtime
gRPC + the Container Runtime Interface

02
The Failure
one message, one hard limit, a node offline

03
The Fix
six streaming RPCs + a rollout trick worth stealing

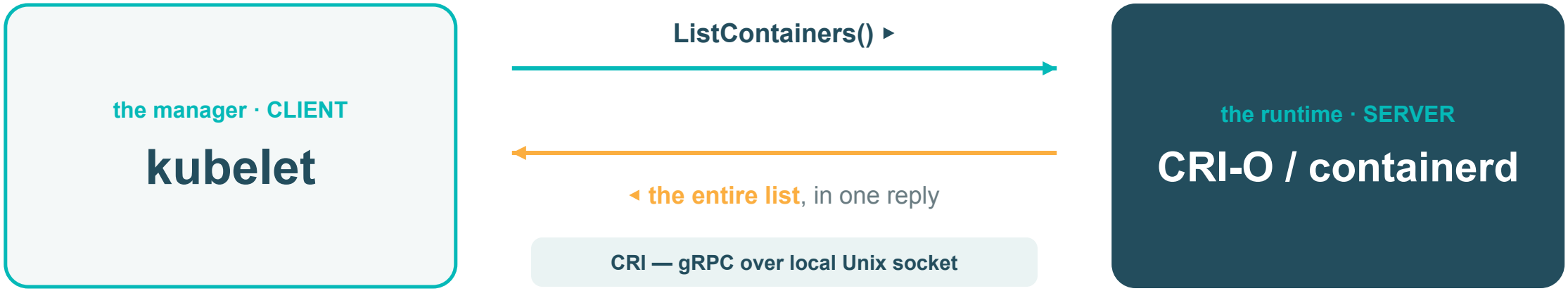
04
The Twist
what streaming solved and what it didn't

05
What It Means
for any gRPC service that returns data at scale

06
Q/A
Questions and answers session

gRPC in Kubernetes: the kubelet ↔ runtime seam

gRPConf NORTH AMERICA



Node Topology

Every Kubernetes node runs both the **kubelet** (manager) and a **container runtime** (worker).

Interface (CRI)

They speak CRI over a local socket. With **43 methods**, the kubelet acts strictly as the client.

The Crucial Limit

The list methods return everything in one massive message.

Hold onto that detail.

It starts with a log line. Twice.

gRPConf NORTH AMERICA

2018

```
ResourceExhausted: received message
larger than max (4195017 vs. 4194304)
```

2025

```
ResourceExhausted: trying to send message
larger than max (16795768 vs. 16777216)
```

4,194,304 = 4 MiB · 16,777,216 = 16 MiB — the kubelet's receive cap.

Same API. Same failure. The limit was doubled twice in between. It broke anyway.

It didn't arrive as a gRPC problem

gRPConf NORTH AMERICA

A team running CI at scale filed a support ticket with five symptoms:

- kubelet errors
- nodes NotReady
- high runtime memory
- slow API server
- database load

Their Actual Ask

“Make the pod cleanup threshold configurable.”

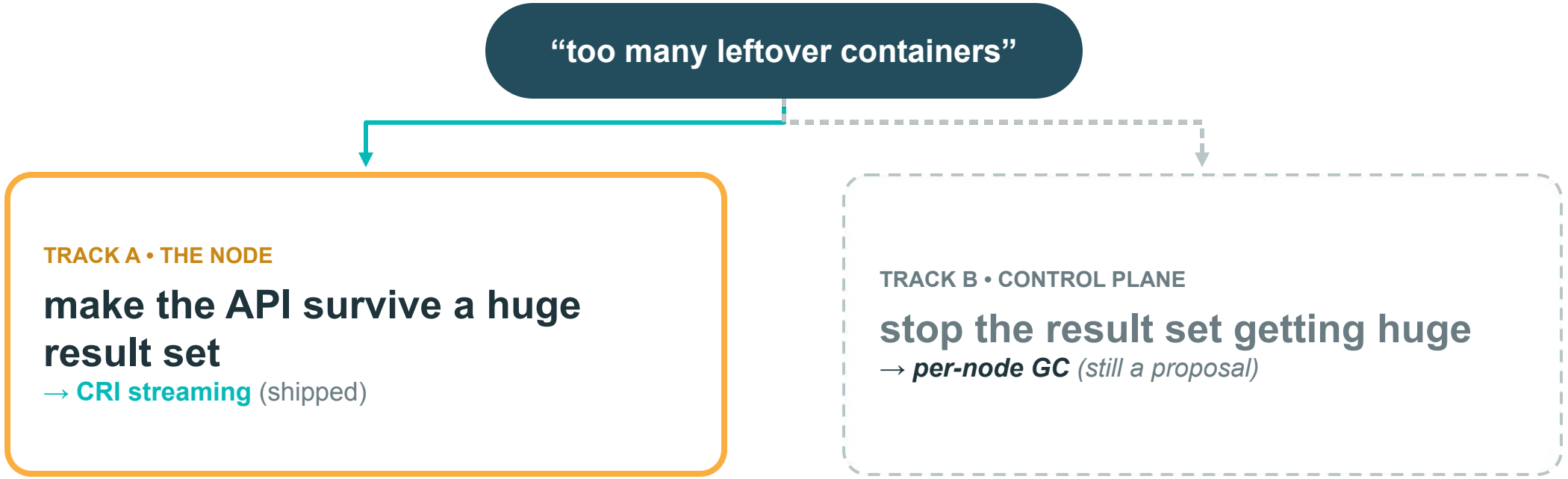
The word “gRPC” does not appear as a problem anywhere in the ticket.

So why am I standing at a gRPC conference?

The wrong question, then the right one

~~✕ Is this a control-plane problem or a node problem?~~

✓ What exactly breaks, and at what threshold?



"A symptom in your component doesn't mean the fix lives there."

What actually breaks


```
rpc ListContainers(ListContainersRequest)
returns (ListContainersResponse)
```

THE DESIGN LIMITS

Returns Everything

No limit, no cursor, and capped at a single 16 MiB message payload limit.

Scale Constraints

Perfect for typical nodes with ~100 containers.

However, container runtimes retain exited containers, causing busy nodes to accumulate thousands.

THE BREAKING POINT

~10,000 CONTAINERS ACCUMULATED

The Response Crosses 16 MiB

When payload limits are exceeded, the API call fails entirely.

NOT TRUNCATED. FAILED.

Why one failed call kills the machine

gRPConf NORTH AMERICA

THE TRIGGER

list "everything" fails

Payload crosses the 16 MiB single-message ceiling.

node → NotReady

STEP 1

Cleanup fails to execute

The garbage collection routine is blocked because it needs the very list that just failed to load.

STEP 2

Nothing is garbage collected

Zero old or exited containers are pruned, locking the system in its bloated state.

STEP 3

More containers pile up

New workloads keep creating containers, driving the list size and payload even further past the limit.

RECOVERY PATH IS BROKEN

The recovery path IS the broken path. The node can’t dig itself out.

“The janitor’s key is locked inside the room he needs to clean.”

The constraint that shaped everything

gRPConf NORTH AMERICA

The API Constraint

CRI v1 is Frozen

You can add new methods, but you **cannot change or remove** existing ones.

The Solution

Six new streaming RPCs introduced alongside the six old ones — leaving the original API completely unmodified.

6 New Streaming RPCs

```
+ StreamContainers
+ StreamPodSandboxes
+ StreamContainerStats
+ StreamPodSandboxStats
+ StreamPodSandboxMetrics
+ StreamImages
```

Most of you aren’t designing new APIs — you’re extending ones you already shipped.

Design for backwards compatibility from day one, because production APIs are forever.

Why pagination lost

THE ORIGINAL PROPOSAL

Pagination KEP

The first design proposed pagination. The legacy directory still carries this name:

```
keps/sig-node/5825-cri-pagination
```

THE FATAL FLAW

Why It Failed

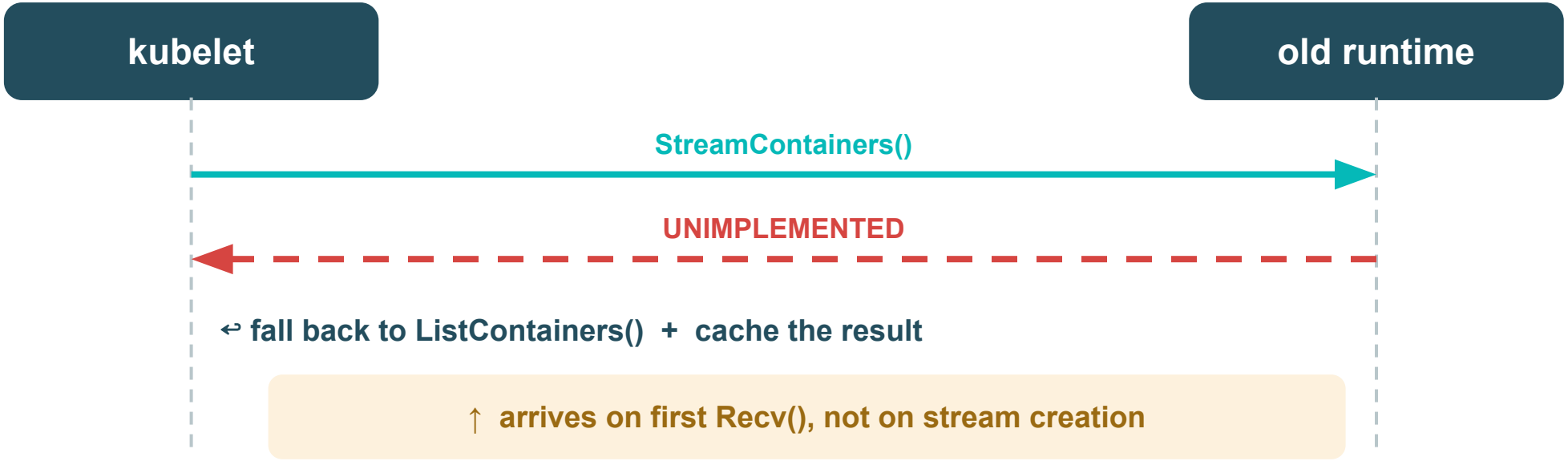
A page token promises a stable order over a container map that mutates every second.

That's a guarantee the store never had.

KEY TAKEAWAY

One traversal is cheaper to guarantee than a stable sort.

The rollout trick worth stealing



- UNIMPLEMENTED is a free capability handshake. No version RPC.
- The latch caching it shipped as a data race → fixed to atomic.Bool. Idempotent still has to be atomic.

Did it work?

Streaming means you never hold the whole thing at once.

So what happened to memory?

Allocations per operation

	allocations / op
unary	480,893
streaming	480,922

difference:
29 allocations

29 objects. Out of half a million.

Streaming changed the wire. It did not change the heap.

Why both ends still buffer

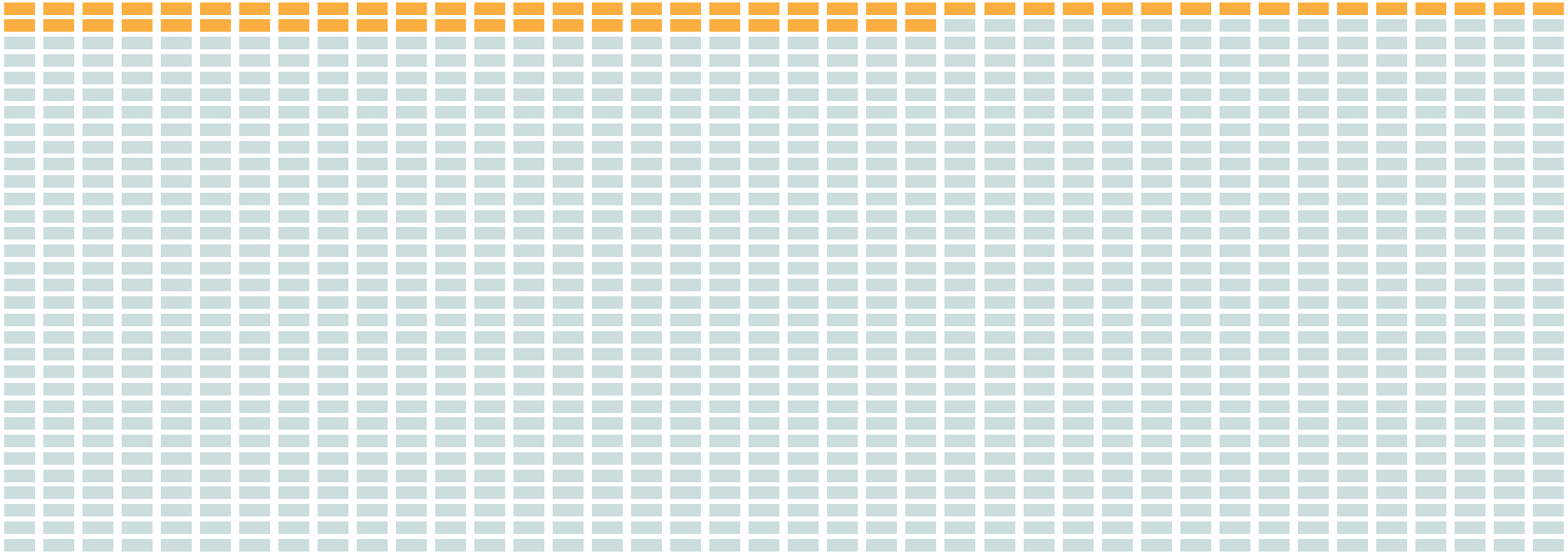

```
// server
ctrs := listContainers(...)
// the WHOLE list, first
for i := 0; i < len(ctrs) {
stream.Send(ctrs[i:end])
}
```

```
// client
for {
resp := stream.Recv()
items = append(items,
resp...)
} // ONE list, then use it
```

You still wrote the entire letter before mailing anything.

Splitting it into envelopes changed the postage — not the paper on your desk.

HTTP/2 was already streaming those bytes



20 MB message ≈ 1,280 DATA frames · 16 KB each · paced across the wire

the 64 amber squares = one flow-control window. Chunking + pacing + backpressure were already there, for the unary call.

The cap is on the reassembled message

gRPConf NORTH AMERICA

01

The 16 MiB Limit

The 16 MiB limit is not a transport limit. It caps how big one reassembled message may be, because a message is an atomic unit of deserialization.

02

The Message Boundary

Streaming moved the message boundary. That's all. It changes how the bytes are paced and assembled across the wire.

A message boundary is a deserialization boundary...

...not an allocation boundary.

What’s still open, and what this was

gRPConf NORTH AMERICA

Both ends still materialize
the whole result set is held in memory

Item count, not byte budget
3,000 fat containers can still bust 16 MiB

No fallback metric
you learn it fell back from a log line

Track B was never built
bounding the count, the original ask

This was a blast door, not a scalability feature.

The goal was to stop a node dying, not to make 10k containers a supported density.

Three things to take home

gRPConf NORTH AMERICA

01

A per-message cap is not a memory bound.

02

An item-count batch is not a byte budget.

03

A streaming method does not inherit your unary interceptor's metrics.

That third one you can check in your own service tonight. Ours doesn't.

gRPConf NORTH AMERICA

Thanks.

Streaming is a **wire-format change** —
not a memory strategy.

KEP-5825

cri-o #9761

containerd #13187



Feedback

Scan to share your thoughts



gRPConf

NORTH AMERICA