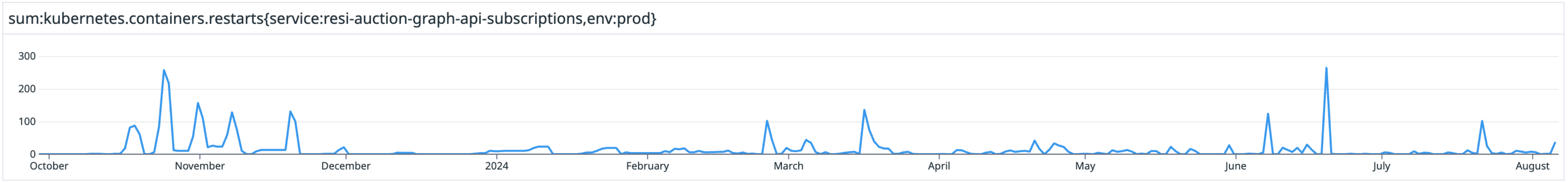
IN MEMORY OF TRAVAILS

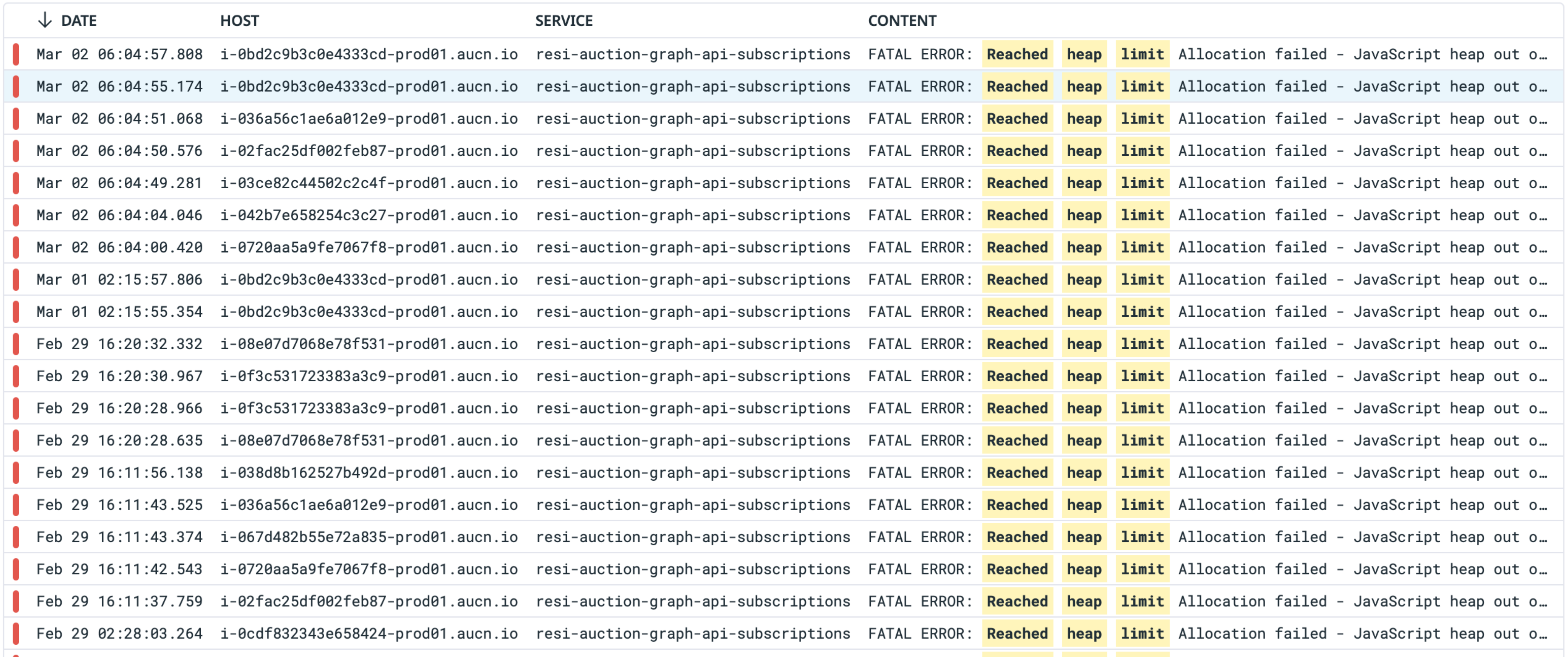
GABRIEL SCHULHOF

AUCTION.COM

BEYOND THE BID.

The Problem





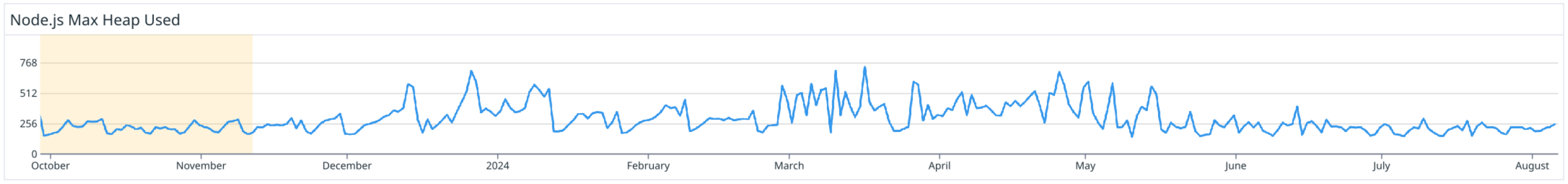
↓ DATE	HOST	SERVICE	CONTENT
Mar 02 06:04:57.808	i-0bd2c9b3c0e4333cd-prod01.aucn.io	resi-auction-graph-api-subscriptions	FATAL ERROR: Reached heap limit Allocation failed - JavaScript heap out o…
Mar 02 06:04:55.174	i-0bd2c9b3c0e4333cd-prod01.aucn.io	resi-auction-graph-api-subscriptions	FATAL ERROR: Reached heap limit Allocation failed - JavaScript heap out o…
Mar 02 06:04:51.068	i-036a56c1ae6a012e9-prod01.aucn.io	resi-auction-graph-api-subscriptions	FATAL ERROR: Reached heap limit Allocation failed - JavaScript heap out o…
Mar 02 06:04:50.576	i-02fac25df002feb87-prod01.aucn.io	resi-auction-graph-api-subscriptions	FATAL ERROR: Reached heap limit Allocation failed - JavaScript heap out o…
Mar 02 06:04:49.281	i-03ce82c44502c2c4f-prod01.aucn.io	resi-auction-graph-api-subscriptions	FATAL ERROR: Reached heap limit Allocation failed - JavaScript heap out o…
Mar 02 06:04:04.046	i-042b7e658254c3c27-prod01.aucn.io	resi-auction-graph-api-subscriptions	FATAL ERROR: Reached heap limit Allocation failed - JavaScript heap out o…
Mar 02 06:04:00.420	i-0720aa5a9fe7067f8-prod01.aucn.io	resi-auction-graph-api-subscriptions	FATAL ERROR: Reached heap limit Allocation failed - JavaScript heap out o…
Mar 01 02:15:57.806	i-0bd2c9b3c0e4333cd-prod01.aucn.io	resi-auction-graph-api-subscriptions	FATAL ERROR: Reached heap limit Allocation failed - JavaScript heap out o…
Mar 01 02:15:55.354	i-0bd2c9b3c0e4333cd-prod01.aucn.io	resi-auction-graph-api-subscriptions	FATAL ERROR: Reached heap limit Allocation failed - JavaScript heap out o…
Feb 29 16:20:32.332	i-08e07d7068e78f531-prod01.aucn.io	resi-auction-graph-api-subscriptions	FATAL ERROR: Reached heap limit Allocation failed - JavaScript heap out o…
Feb 29 16:20:30.967	i-0f3c531723383a3c9-prod01.aucn.io	resi-auction-graph-api-subscriptions	FATAL ERROR: Reached heap limit Allocation failed - JavaScript heap out o…
Feb 29 16:20:28.966	i-0f3c531723383a3c9-prod01.aucn.io	resi-auction-graph-api-subscriptions	FATAL ERROR: Reached heap limit Allocation failed - JavaScript heap out o…
Feb 29 16:20:28.635	i-08e07d7068e78f531-prod01.aucn.io	resi-auction-graph-api-subscriptions	FATAL ERROR: Reached heap limit Allocation failed - JavaScript heap out o…
Feb 29 16:11:56.138	i-038d8b162527b492d-prod01.aucn.io	resi-auction-graph-api-subscriptions	FATAL ERROR: Reached heap limit Allocation failed - JavaScript heap out o…
Feb 29 16:11:43.525	i-036a56c1ae6a012e9-prod01.aucn.io	resi-auction-graph-api-subscriptions	FATAL ERROR: Reached heap limit Allocation failed - JavaScript heap out o…
Feb 29 16:11:43.374	i-067d482b55e72a835-prod01.aucn.io	resi-auction-graph-api-subscriptions	FATAL ERROR: Reached heap limit Allocation failed - JavaScript heap out o…
Feb 29 16:11:42.543	i-0720aa5a9fe7067f8-prod01.aucn.io	resi-auction-graph-api-subscriptions	FATAL ERROR: Reached heap limit Allocation failed - JavaScript heap out o…
Feb 29 16:11:37.759	i-02fac25df002feb87-prod01.aucn.io	resi-auction-graph-api-subscriptions	FATAL ERROR: Reached heap limit Allocation failed - JavaScript heap out o…
Feb 29 02:28:03.264	i-0cdf832343e658424-prod01.aucn.io	resi-auction-graph-api-subscriptions	FATAL ERROR: Reached heap limit Allocation failed - JavaScript heap out o…

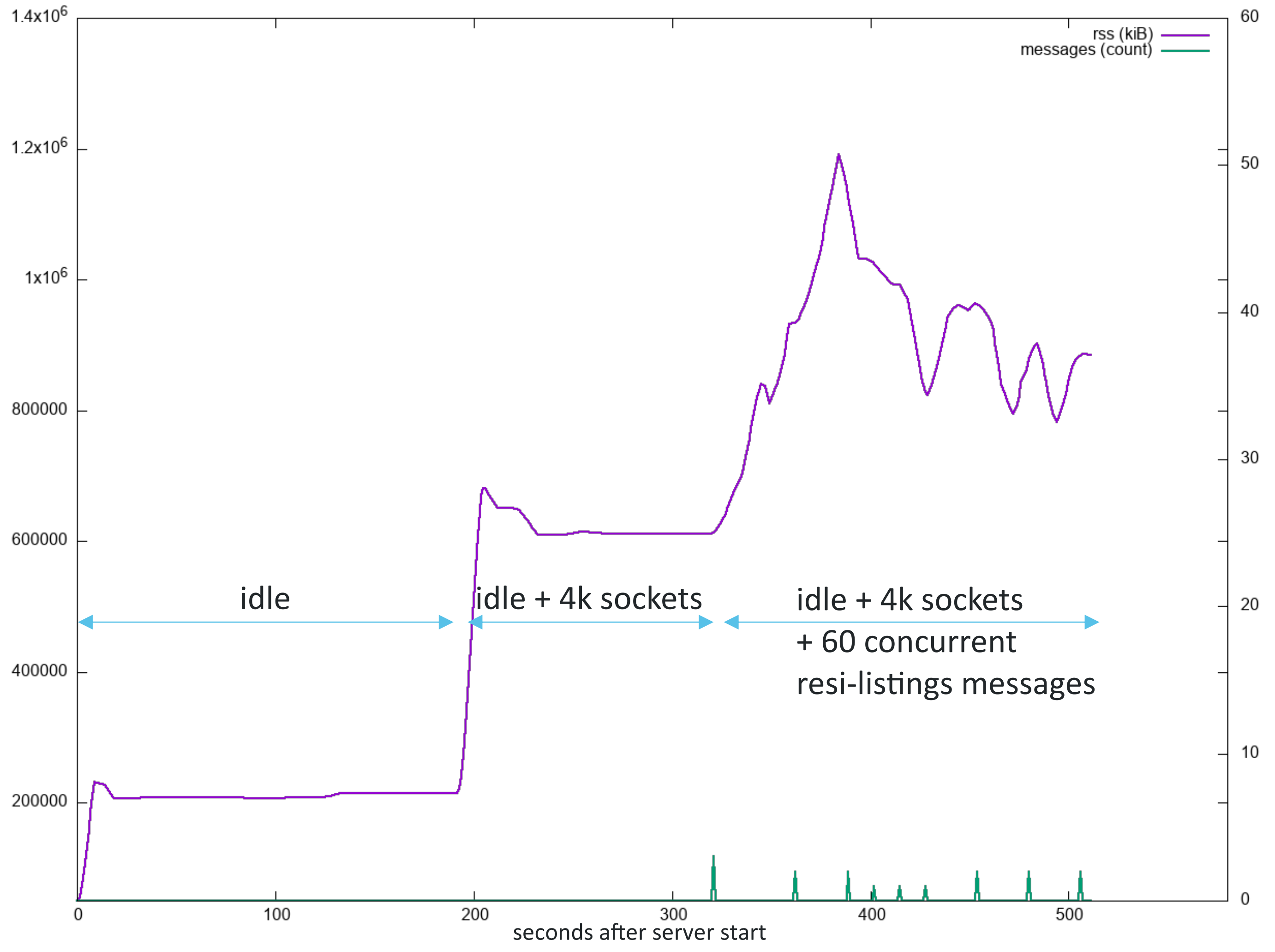
The Setup

Background
- Graph subscribes to kafka topics, uses graphql-subscriptions-redis for pubsub

To Test (on a Mac laptop):
- Run kafka locally (zookeeper + broker)
- Point graph to local broker
- Connect 4000 Websockets
- Use **`kcat`** in several concurrent shell loops to flood the broker with many copies of a single message.

Baseline

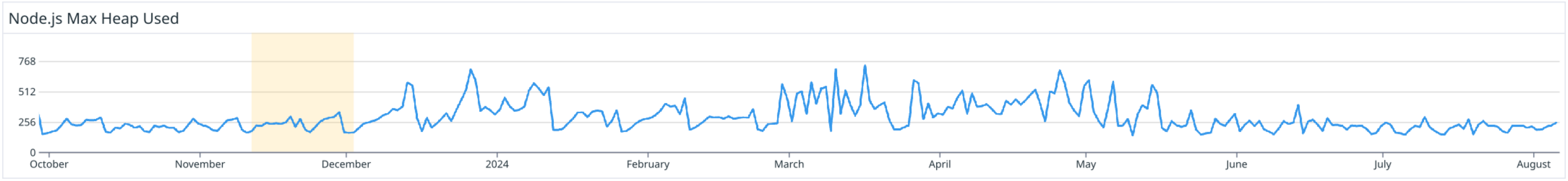




Message Counts (baseline):

```
recv:      16
send:   56000
```

Lazify Backends



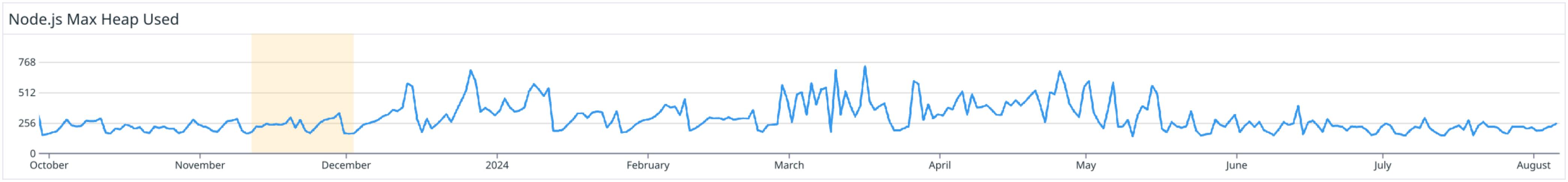
- We have 32 backends.
- Definition depends on context ➔One set per query/socket.
- Almost no socket uses any of them.
- No query uses all 32.
- …
- Q: Why create all of them for every query and every socket?
  A: No good reason. Create them as-needed.

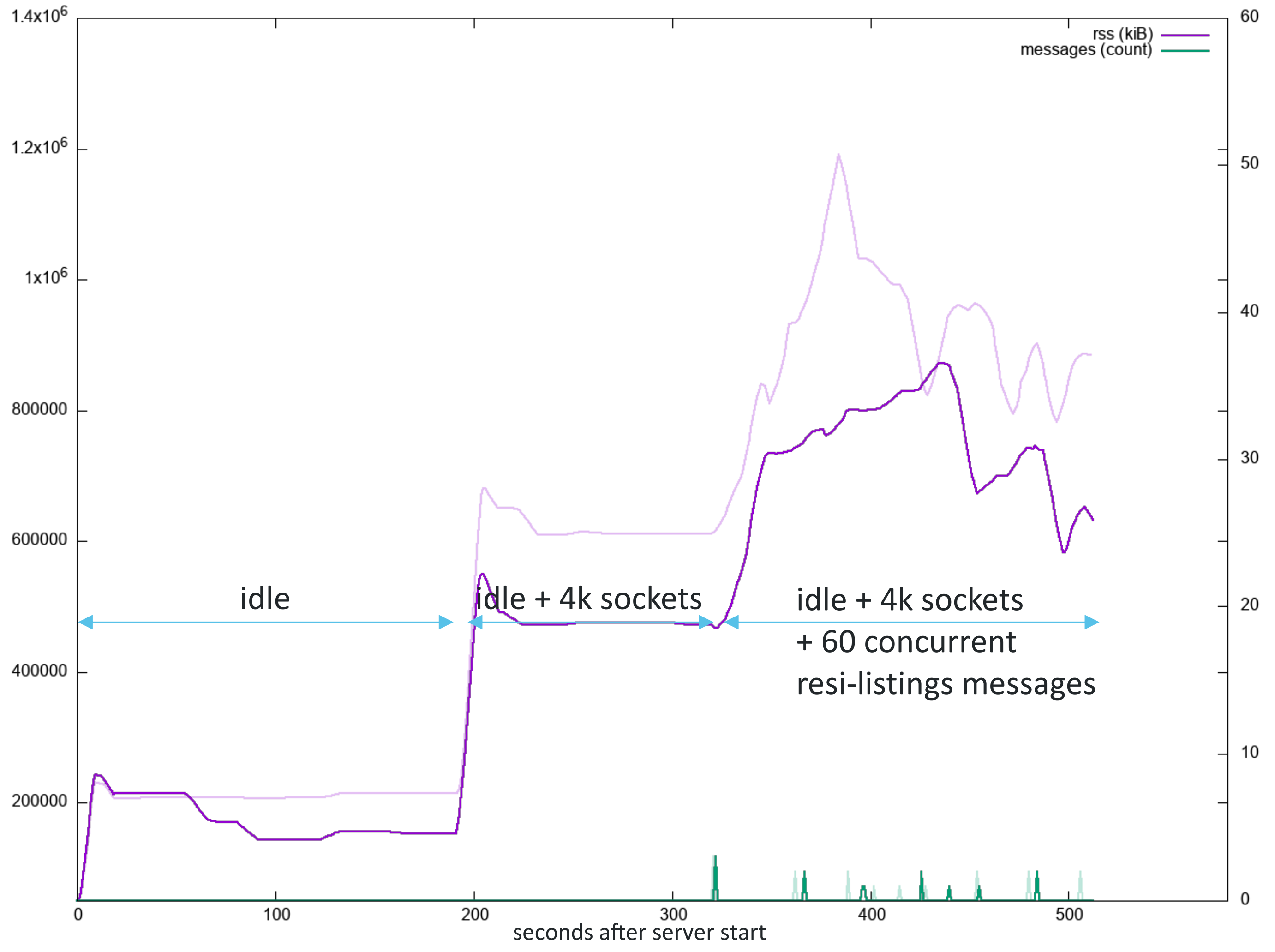
```
new Proxy({}, {
  get: (target, propName) => {
    return target[propertyName] ||
      (target[propertyName] =
        createBackend(ctx, propName))
  }
})
…
ctx.backends.uaa.post('/…', …)
```

auction	poke
audit	potentialreturn
consumerListingIntake	preferences
contract	profile
document	property
geography	riddler
gls	salesforce
intakeStandardization	seek
listing	seller
mlhintegrator	sellerdashboard
morphlog	tenflix
notify	tracking
offer	trinity
onlineAuction	uaa
partysearch	vendor
payment	venue

The look and feel of code that uses the backends was unchanged 🎉

Lazify Backends

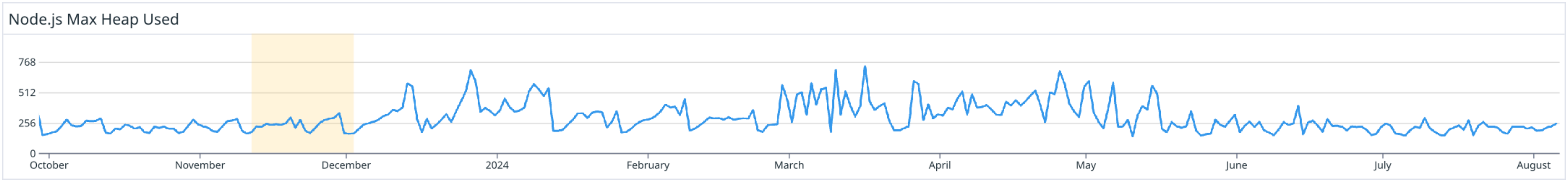


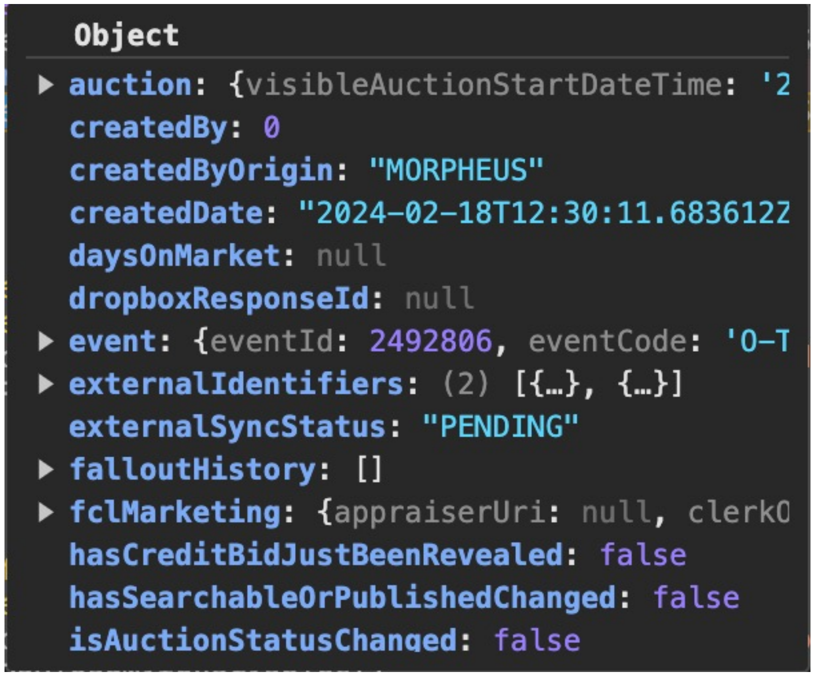


Message Counts (vs. baseline):

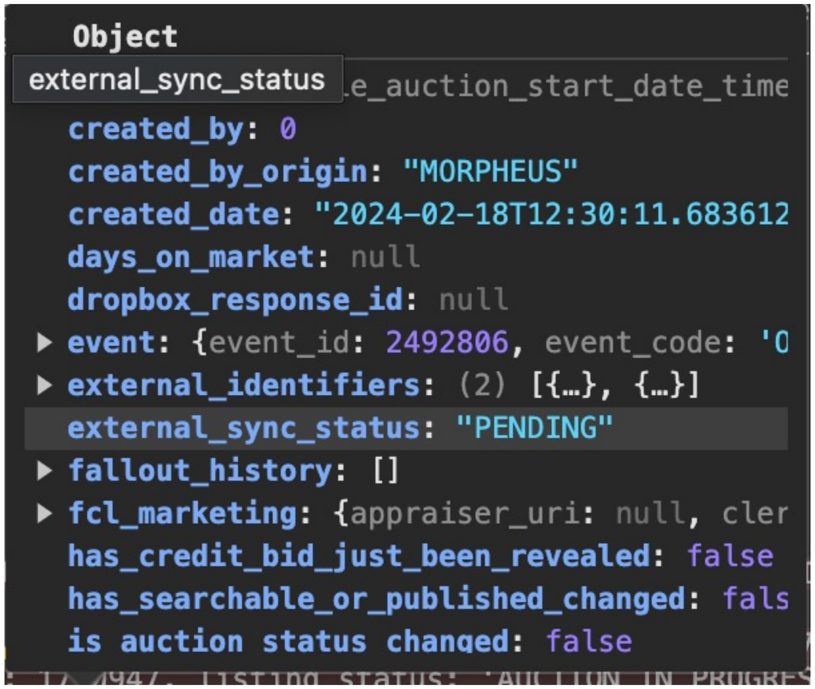
```
recv:     13 ▼  18.8%
send:  52000 ▼   7.1%
```

Speed up convertObjToSnakeKeys





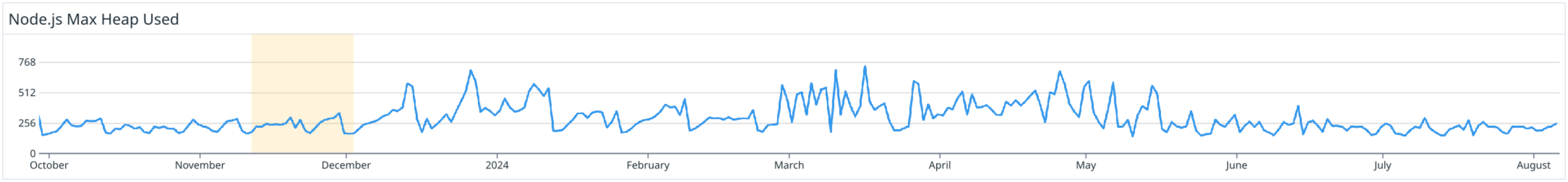
```
Object
▶ auction: {visibleAuctionStartDateTime: '2
  createdBy: 0
  createdByOrigin: "MORPHEUS"
  createdDate: "2024-02-18T12:30:11.683612Z
  daysOnMarket: null
  dropboxResponseId: null
▶ event: {eventId: 2492806, eventCode: 'O-T
▶ externalIdentifiers: (2) [{…}, {…}]
  externalSyncStatus: "PENDING"
▶ falloutHistory: []
▶ fclMarketing: {appraiserUri: null, clerkO
  hasCreditBidJustBeenRevealed: false
  hasSearchableOrPublishedChanged: false
  isAuctionStatusChanged: false
```

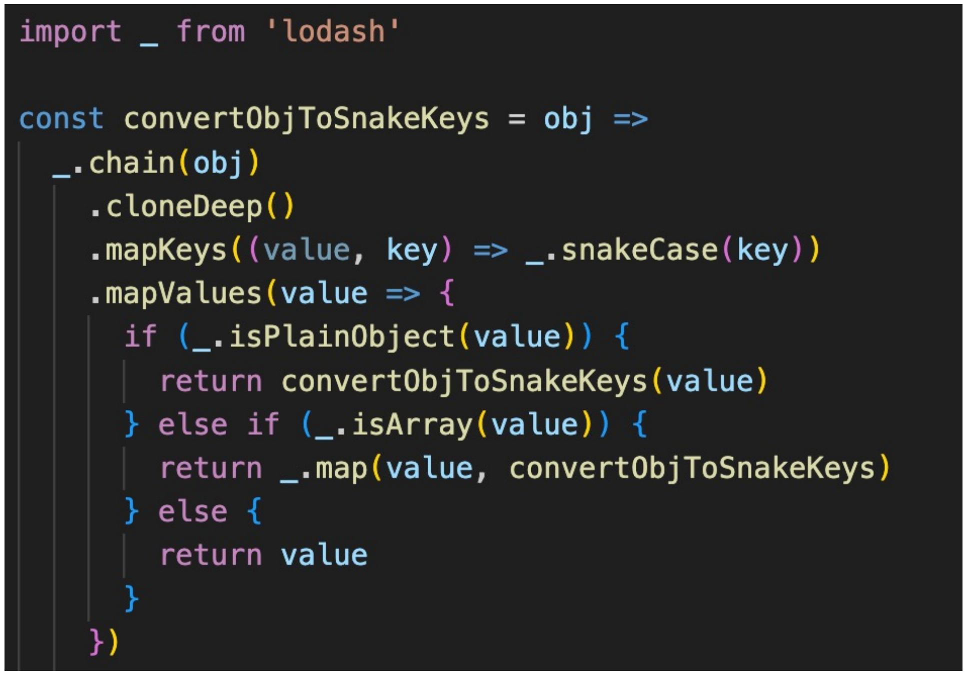


```
Object
external_sync_status e_auction_start_date_time
  created_by: 0
  created_by_origin: "MORPHEUS"
  created_date: "2024-02-18T12:30:11.683612
  days_on_market: null
  dropbox_response_id: null
▶ event: {event_id: 2492806, event_code: 'O
▶ external_identifiers: (2) [{…}, {…}]
  external_sync_status: "PENDING"
▶ fallout_history: []
▶ fcl_marketing: {appraiser_uri: null, cler
  has_credit_bid_just_been_revealed: false
  has_searchable_or_published_changed: fals
  is_auction_status_changed: false
```

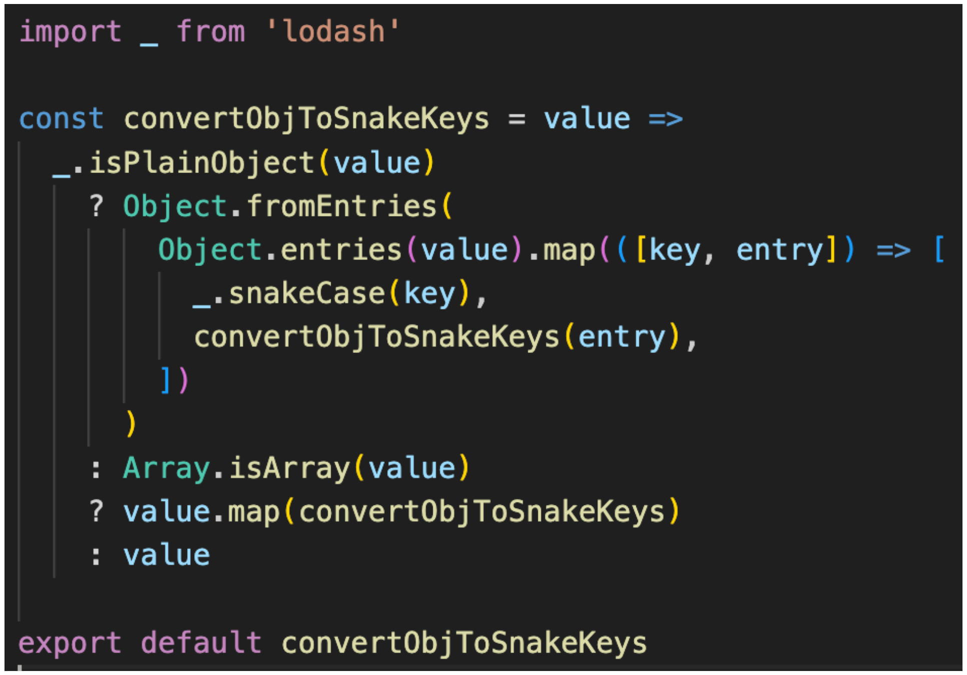
- Rely less on lodash, and more on native iteration.

Speed up convertObjToSnakeKeys





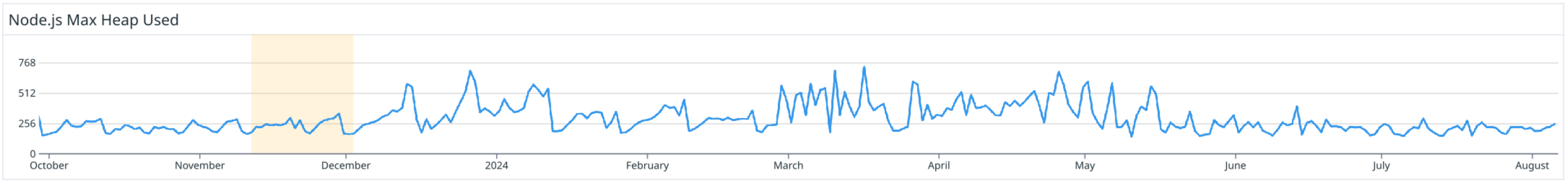
```
import _ from 'lodash'

const convertObjToSnakeKeys = obj =>
  _.chain(obj)
    .cloneDeep()
    .mapKeys((value, key) => _.snakeCase(key))
    .mapValues(value => {
      if (_.isPlainObject(value)) {
        return convertObjToSnakeKeys(value)
      } else if (_.isArray(value)) {
        return _.map(value, convertObjToSnakeKeys)
      } else {
        return value
      }
    })
```

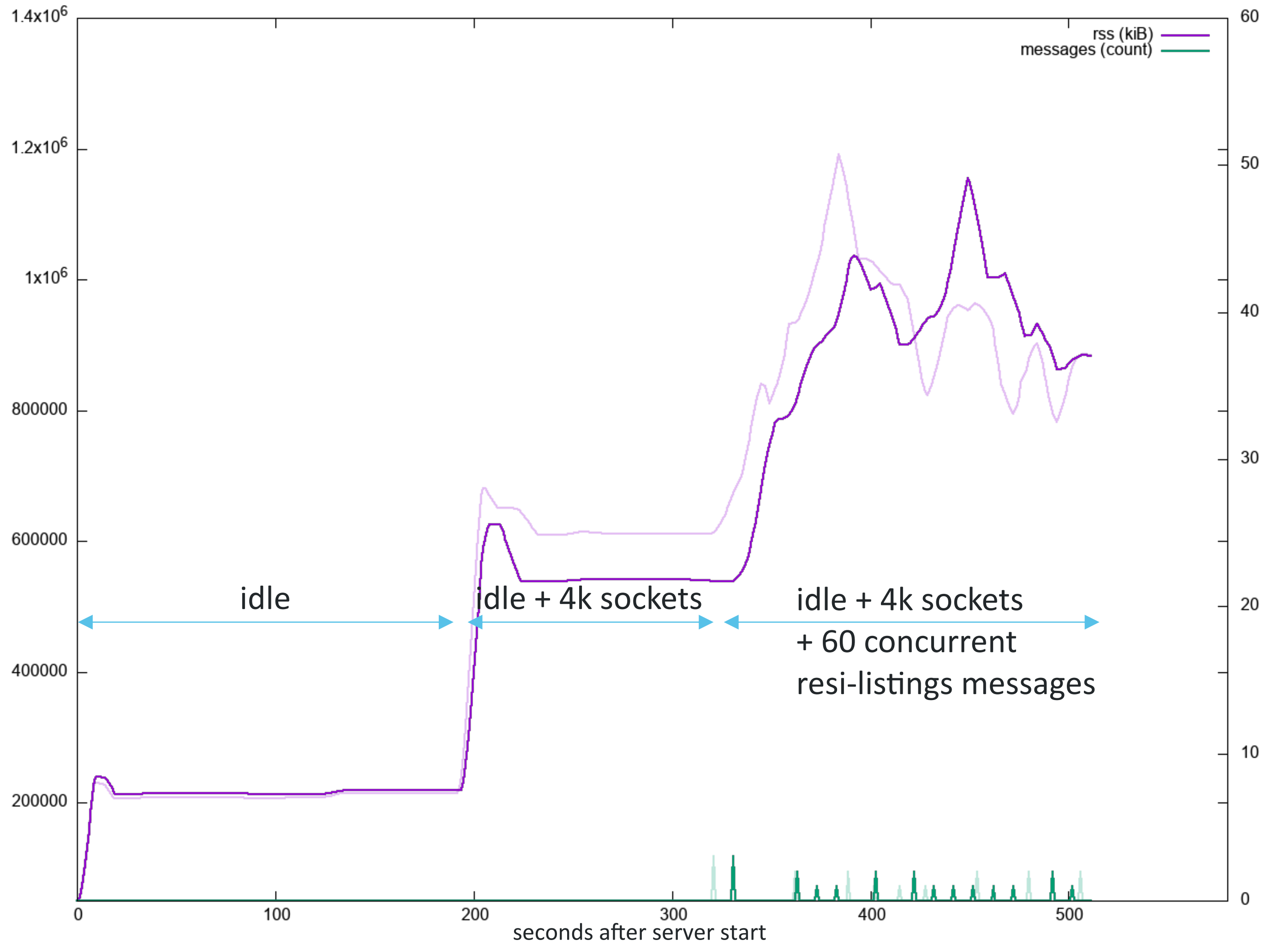


```
import _ from 'lodash'

const convertObjToSnakeKeys = value =>
  _.isPlainObject(value)
    ? Object.fromEntries(
        Object.entries(value).map(([key, entry]) => [
          _.snakeCase(key),
          convertObjToSnakeKeys(entry),
        ])
      )
    : Array.isArray(value)
    ? value.map(convertObjToSnakeKeys)
    : value

export default convertObjToSnakeKeys
```

- Rely less on lodash, and more on native iteration.

Speed up convertObjToSnakeKeys



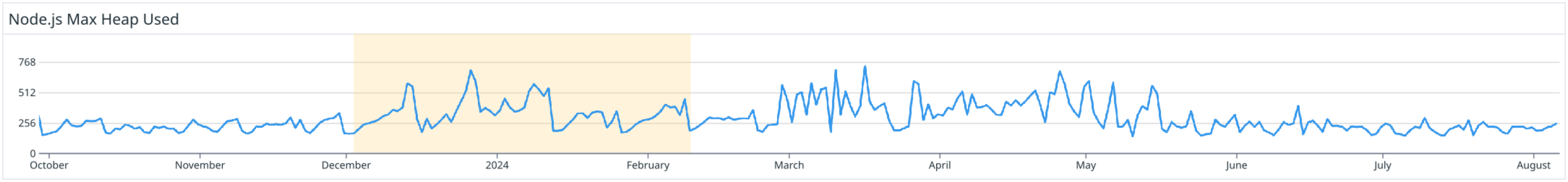


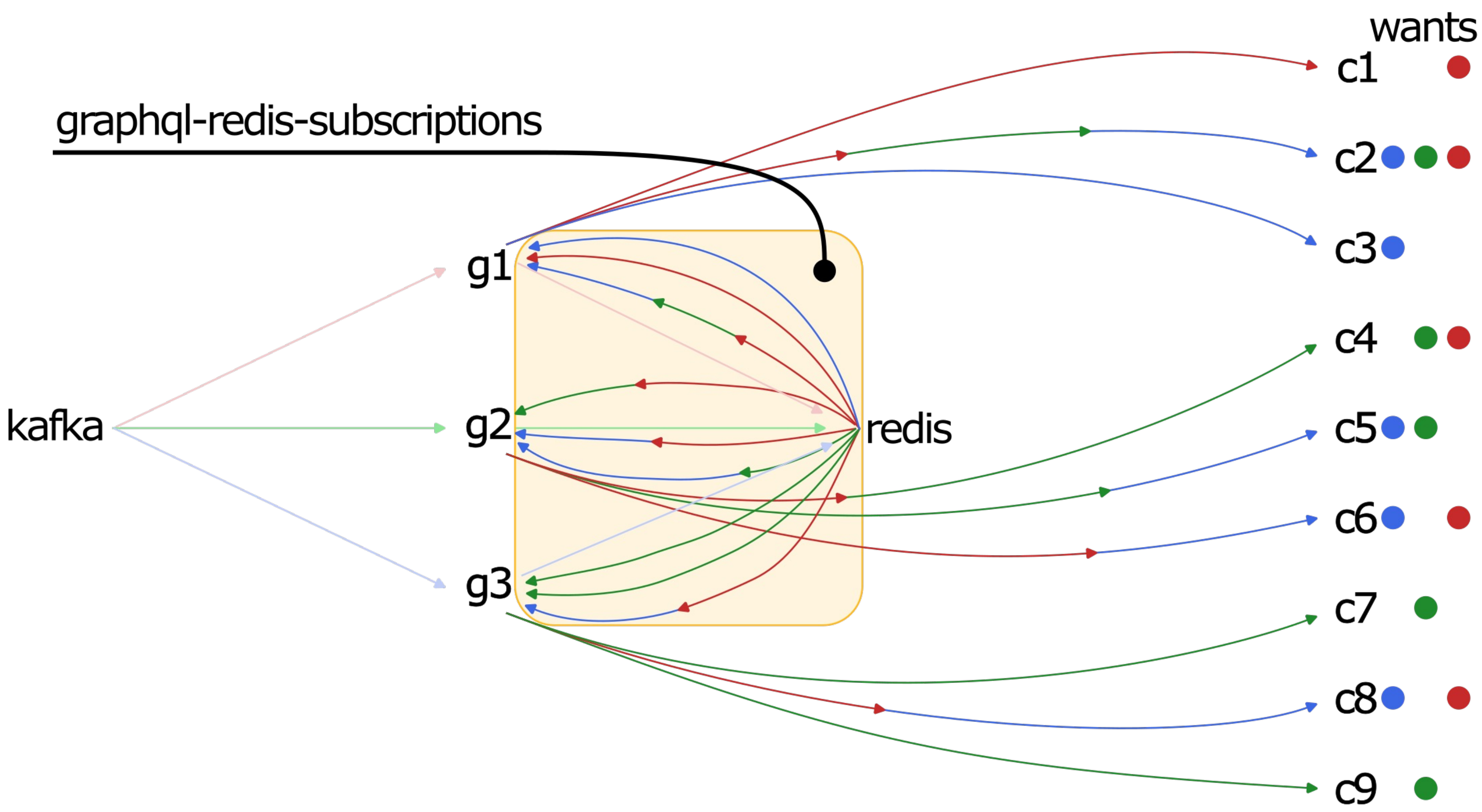
Message Counts (vs. baseline):

```
recv:      19 ▲ 18.8%
send:   72000 ▲ 28.6%
```

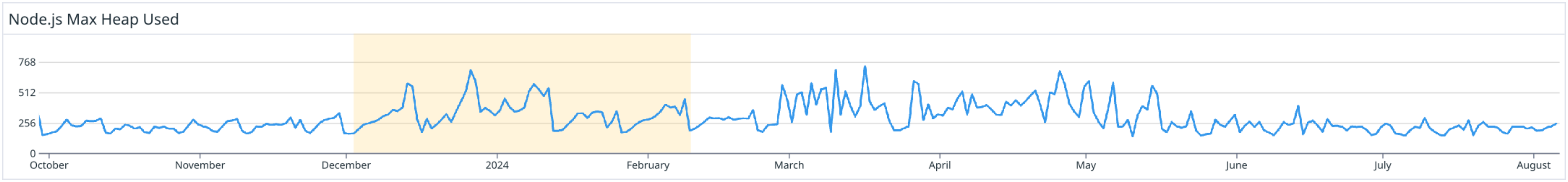
AUCTION.COM BEYOND THE BID.

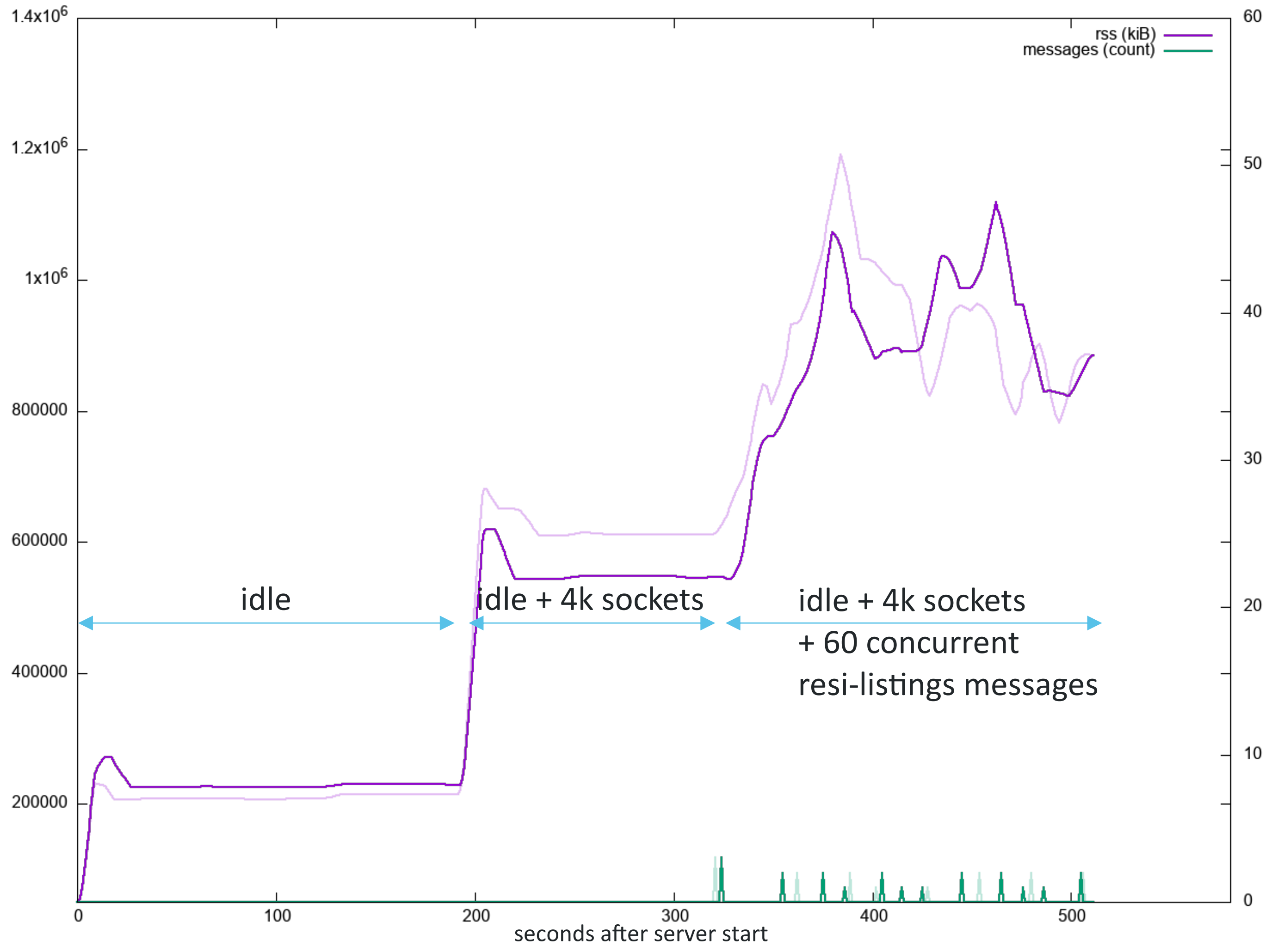
Upgrade graphql-redis-subscriptions





Upgrade graphql-redis-subscriptions

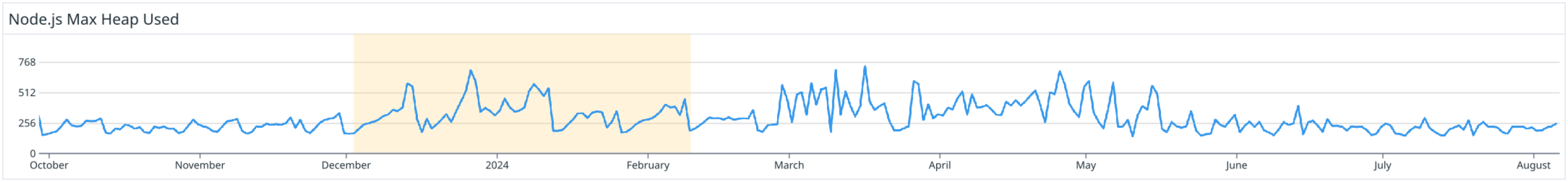


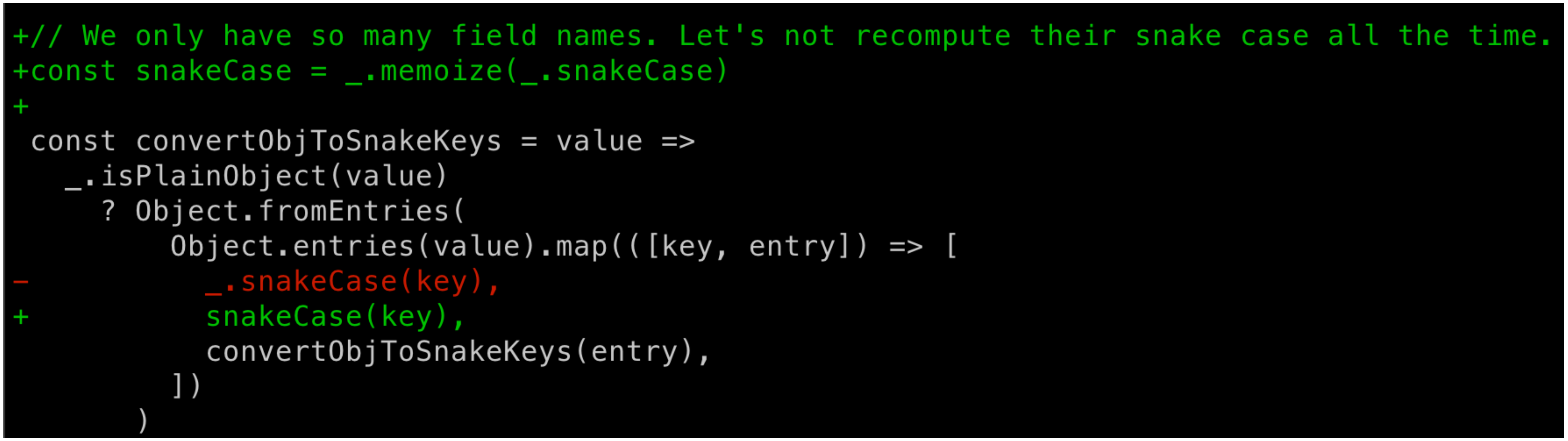


Message Counts (vs. baseline):

```
recv:      20 ▲  25.0%
send:   72000 ▲  28.6%
```

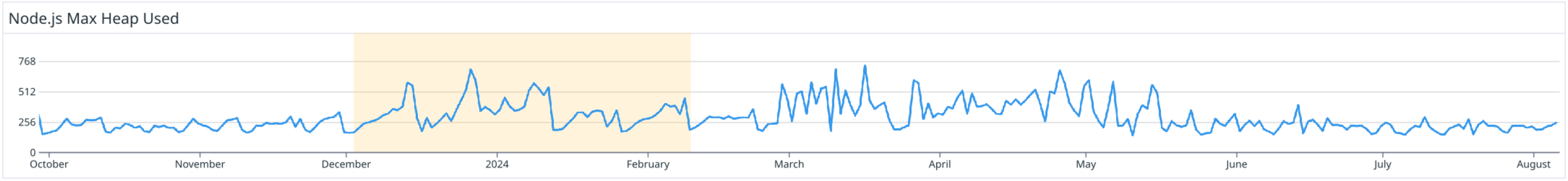
Memoize snakeCase

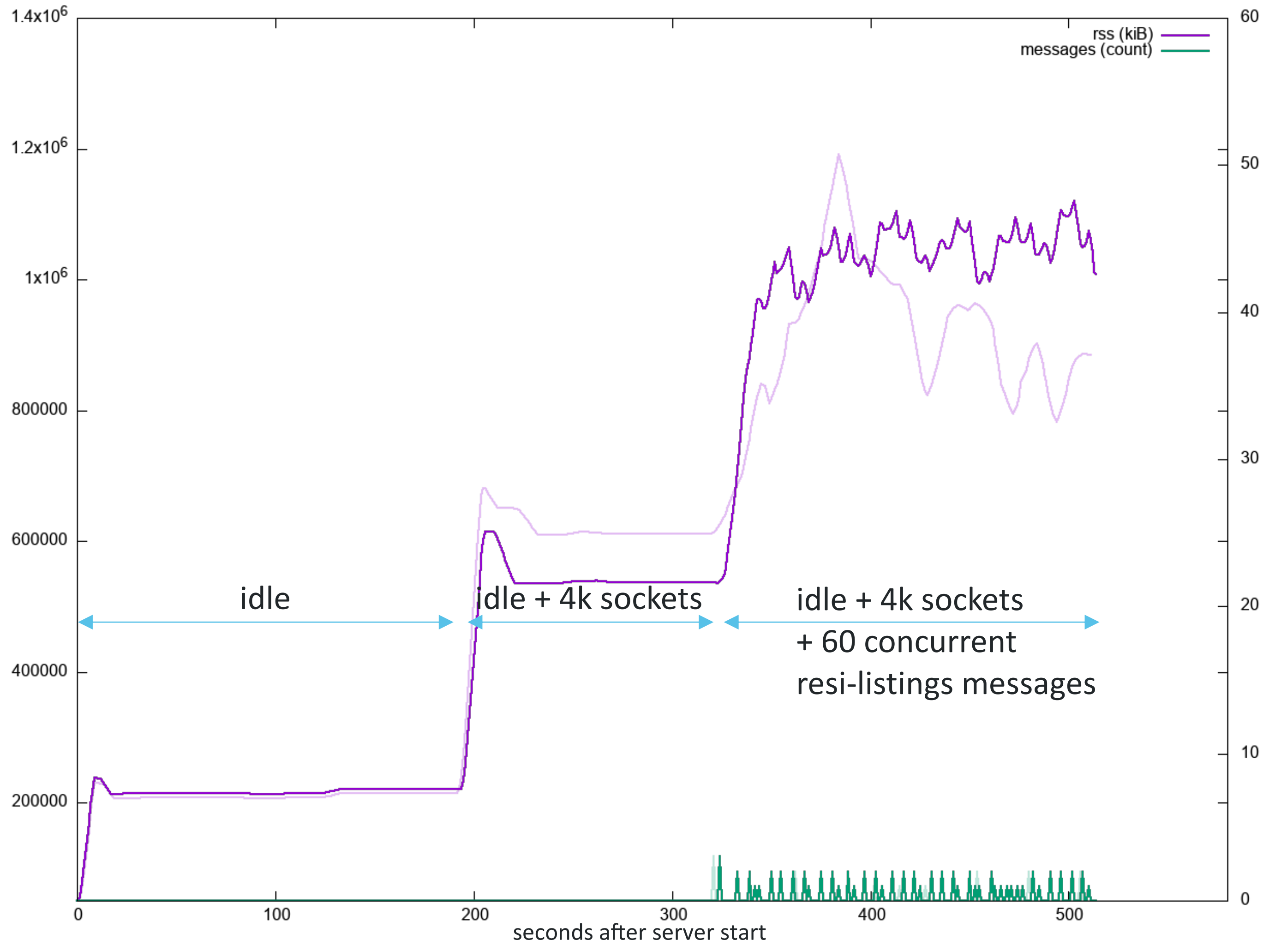




```
+// We only have so many field names. Let's not recompute their snake case all the time.
+const snakeCase = _.memoize(_.snakeCase)
+
 const convertObjToSnakeKeys = value =>
   _.isPlainObject(value)
     ? Object.fromEntries(
         Object.entries(value).map(([key, entry]) => [
-          _.snakeCase(key),
+          snakeCase(key),
           convertObjToSnakeKeys(entry),
         ])
       )
```

Memoize snakeCase

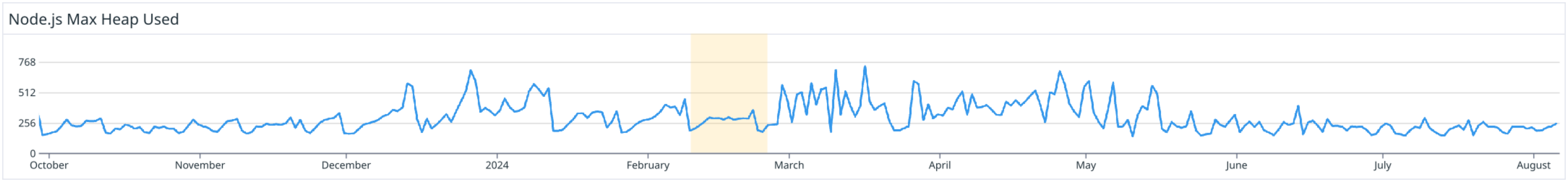




Message Counts (vs. baseline):

```
recv:      69 ▲ 3.313x
send:  276000 ▲ 3.929x
```

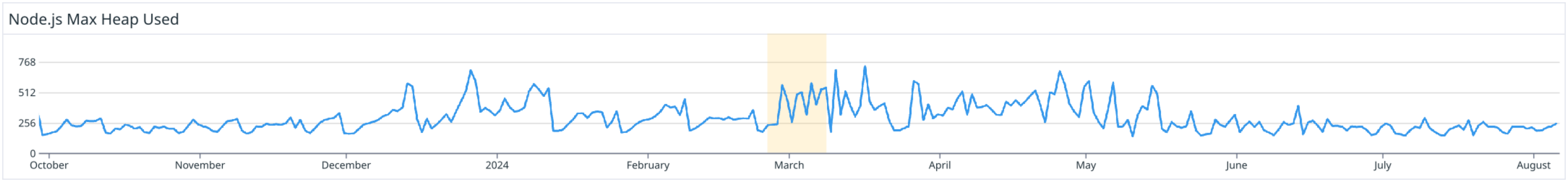
HPA (Horizontal Pod Autoscaling)



Start more pods when needed:

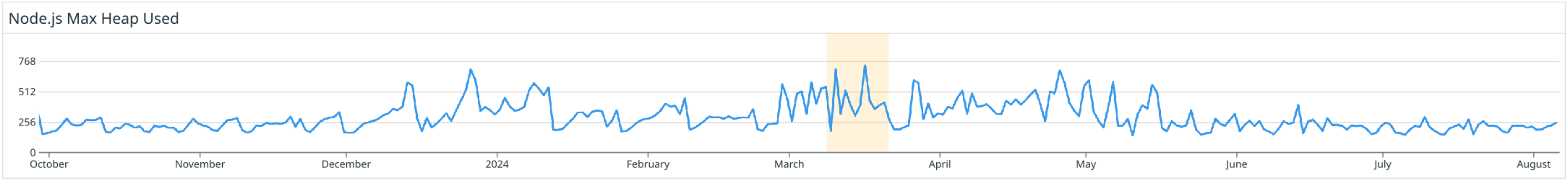
- Avoid restarts
- Memory leak persists
- HPA keyed on memory consumption
- K8s removes the pods it added, not the oldest pods
- Sudden spikes in incoming messages cause memory usage spikes

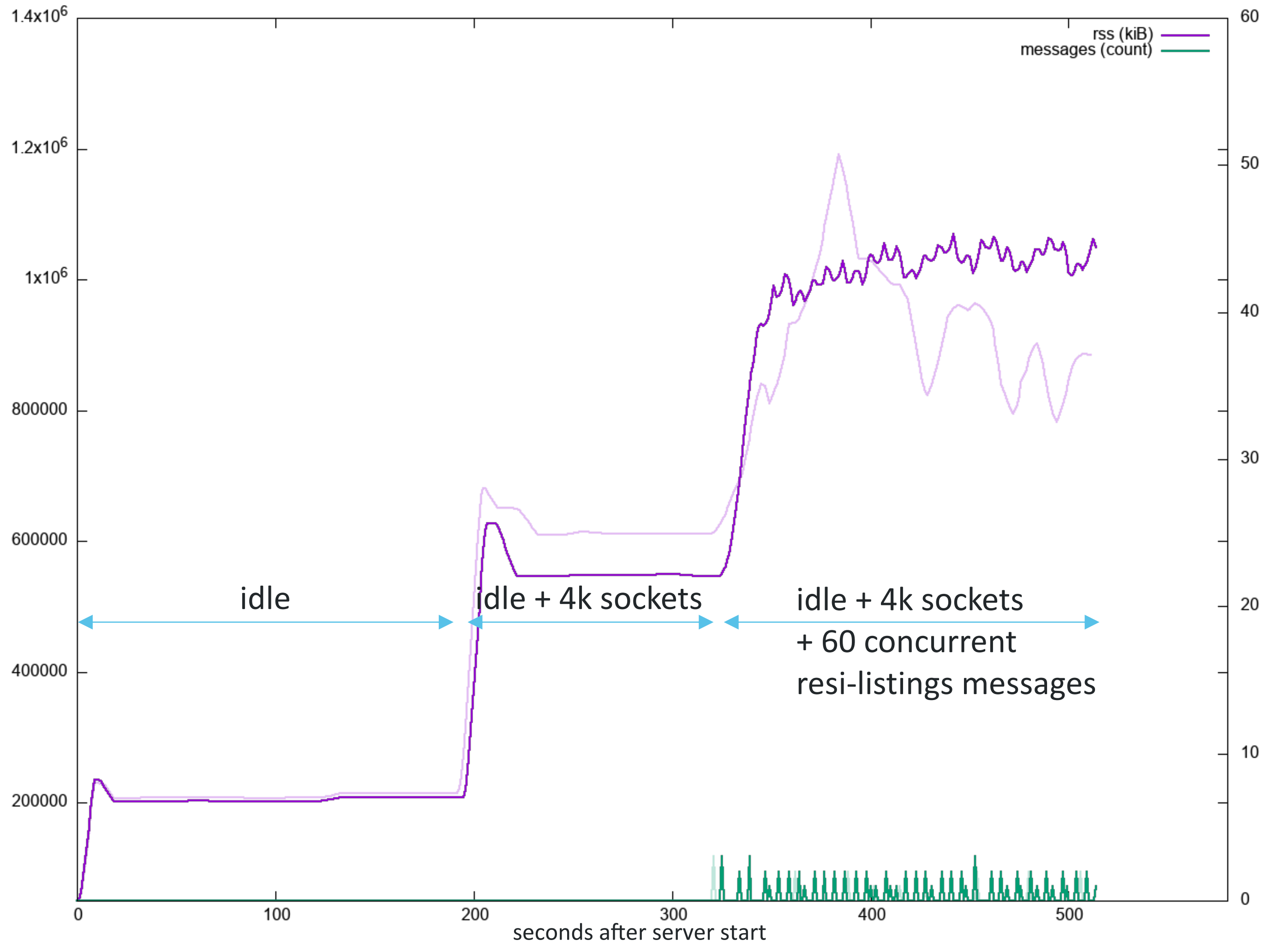
Nightly Restarts



- “Address” memory leaks 🤷‍♂️

Generate Node.js JS





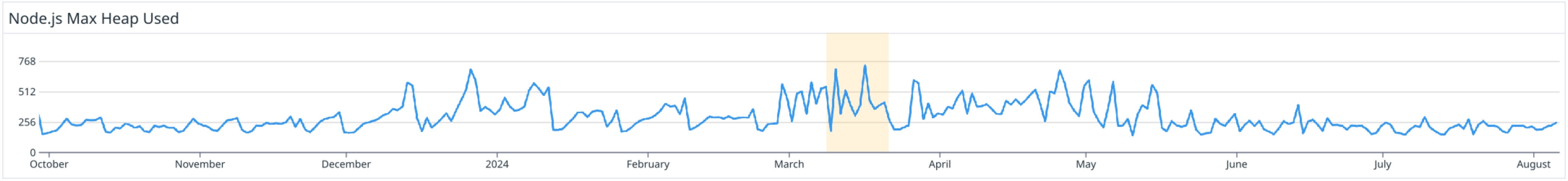
- Build step (babel + webpack)
- ts →js
- js →js
- Browser lowest common denominator
- No spread operator
- No optional chaining
- No async/await
- We control the platform (Node.js 20)

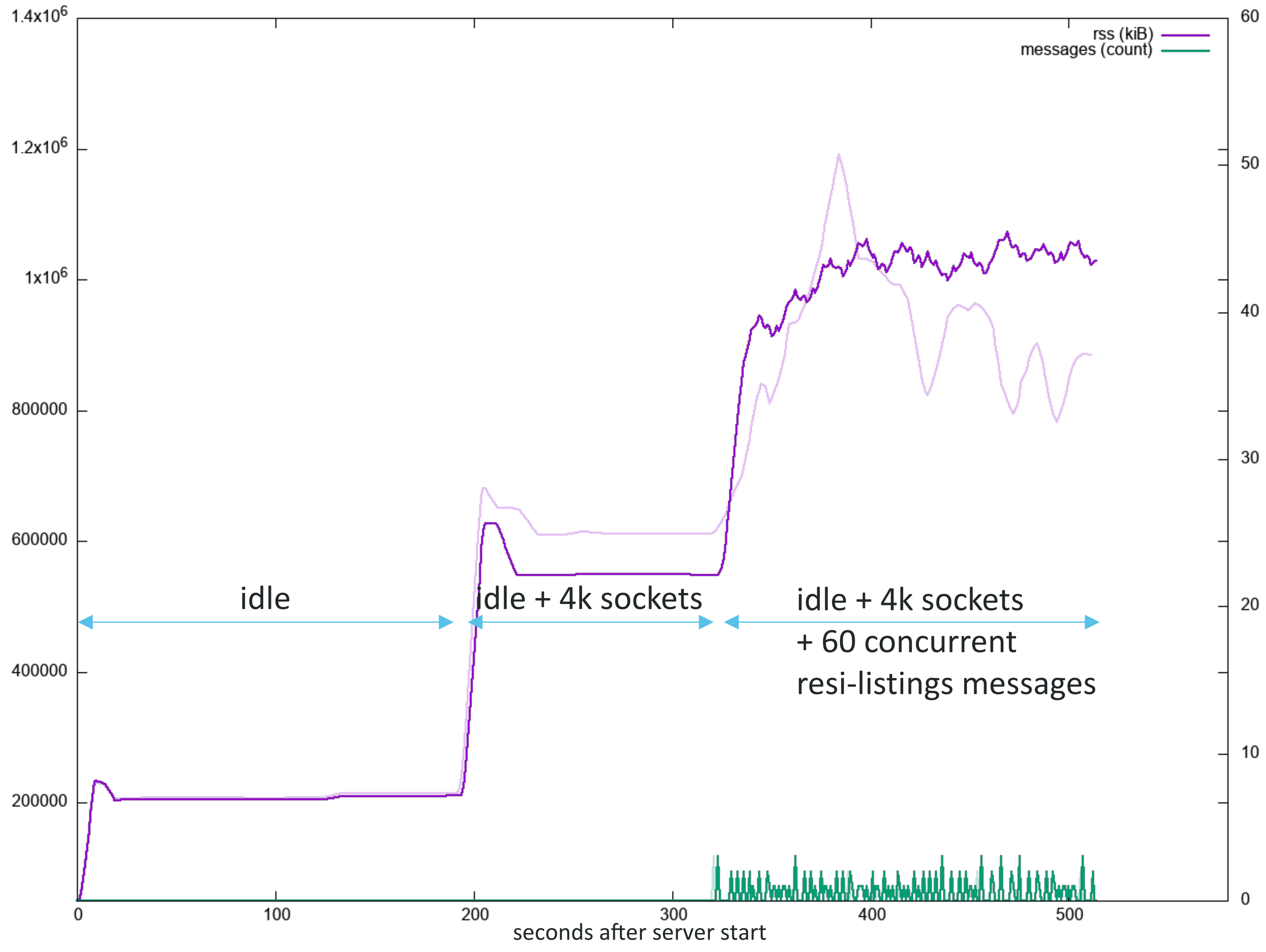
Message Counts (vs. baseline):

```
recv:      75 ▲ 3.688x
send:  300000 ▲ 4.357x
```

AUCTION.COM BEYOND THE BID.

convertObjToSnakeKeys: Do More Ops in Place





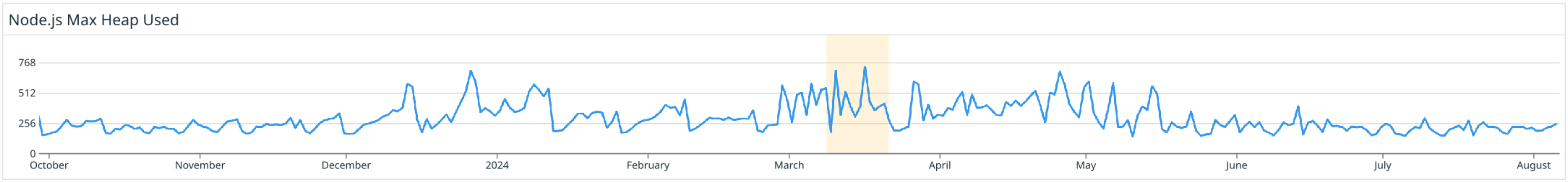
- Replace **`Object.entries`**, **`Object.fromEntries`**, and **`map`** with for loops.
- Create temporary objects only for **{ }** because the keys need to be updated.
- Modify **[]** in place because only values need to be updated.

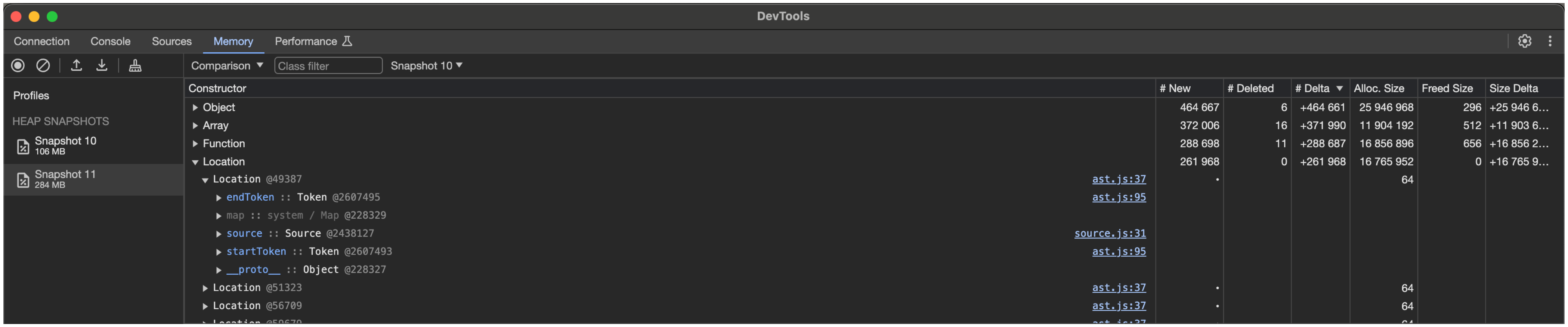
Message Counts (vs. baseline):

```
recv:     117 ▲ 6.312x
send:  476000 ▲ 7.500x
```

AUCTION.COM
BEYOND THE BID.

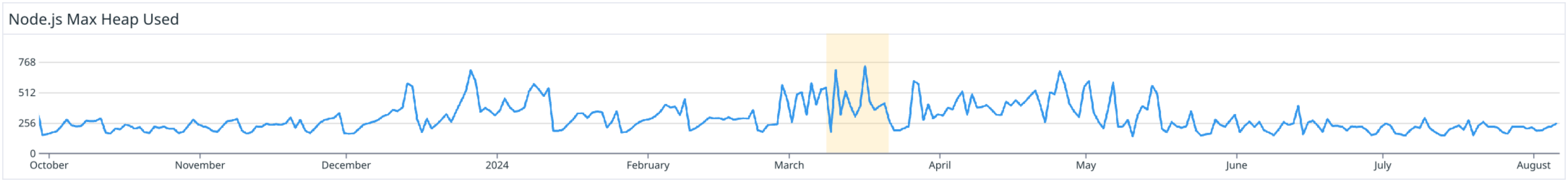
No `Location` in Subscription AST



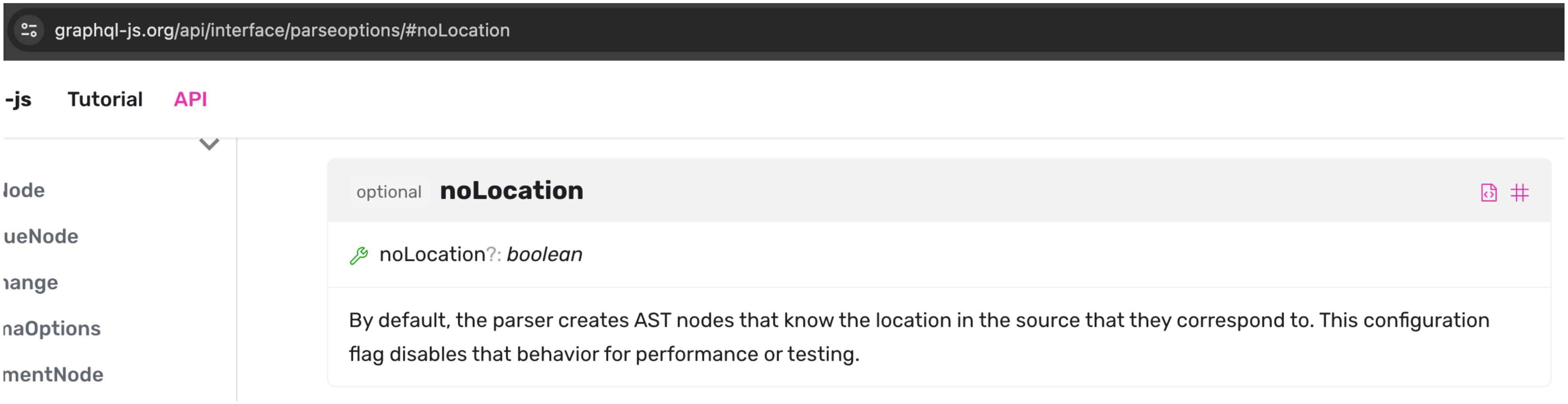


Constructor		# New	# Deleted	# Delta	Alloc. Size	Freed Size	Size Delta
▶ Object		464 667	6	+464 661	25 946 968	296	+25 946 6…
▶ Array		372 006	16	+371 990	11 904 192	512	+11 903 6…
▶ Function		288 698	11	+288 687	16 856 896	656	+16 856 2…
▼ Location		261 968	0	+261 968	16 765 952	0	+16 765 9…
▼ `Location @49387`	`ast.js:37`	•			64		
▶ `endToken :: Token @2607495`	`ast.js:95`						
▶ `map :: system / Map @228329`							
▶ `source :: Source @2438127`	`source.js:31`						
▶ `startToken :: Token @2607493`	`ast.js:95`						
▶ `__proto__ :: Object @228327`							
▶ `Location @51323`	`ast.js:37`	•			64		
▶ `Location @56709`	`ast.js:37`	•			64		

No `Location` in Subscription AST



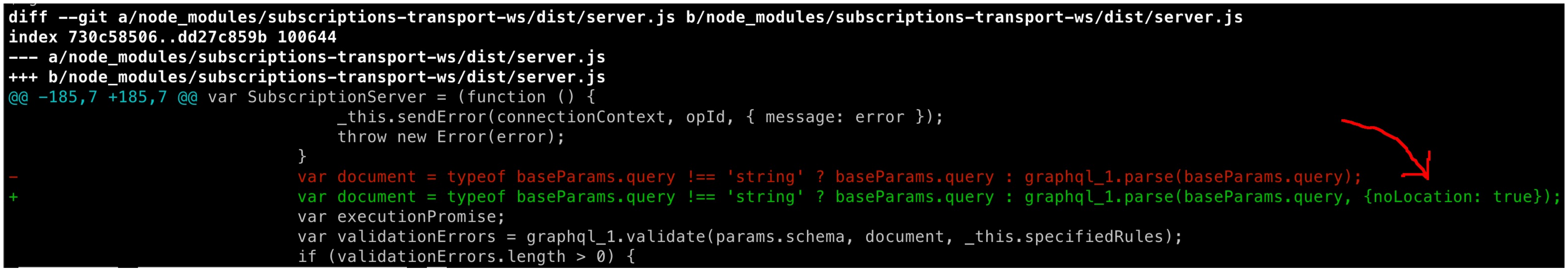
- websocket server parses subscription request
- graphql produces AST (100s of objects for each websocket)
- Each AST node contains a **`Location`** object 🤦‍♂️



optional **noLocation**

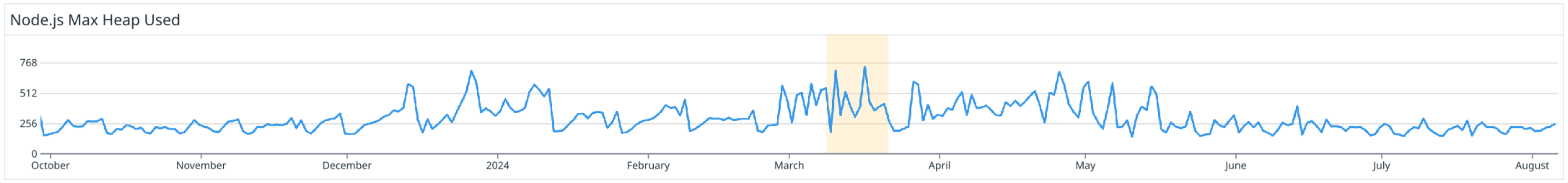
noLocation?: *boolean*

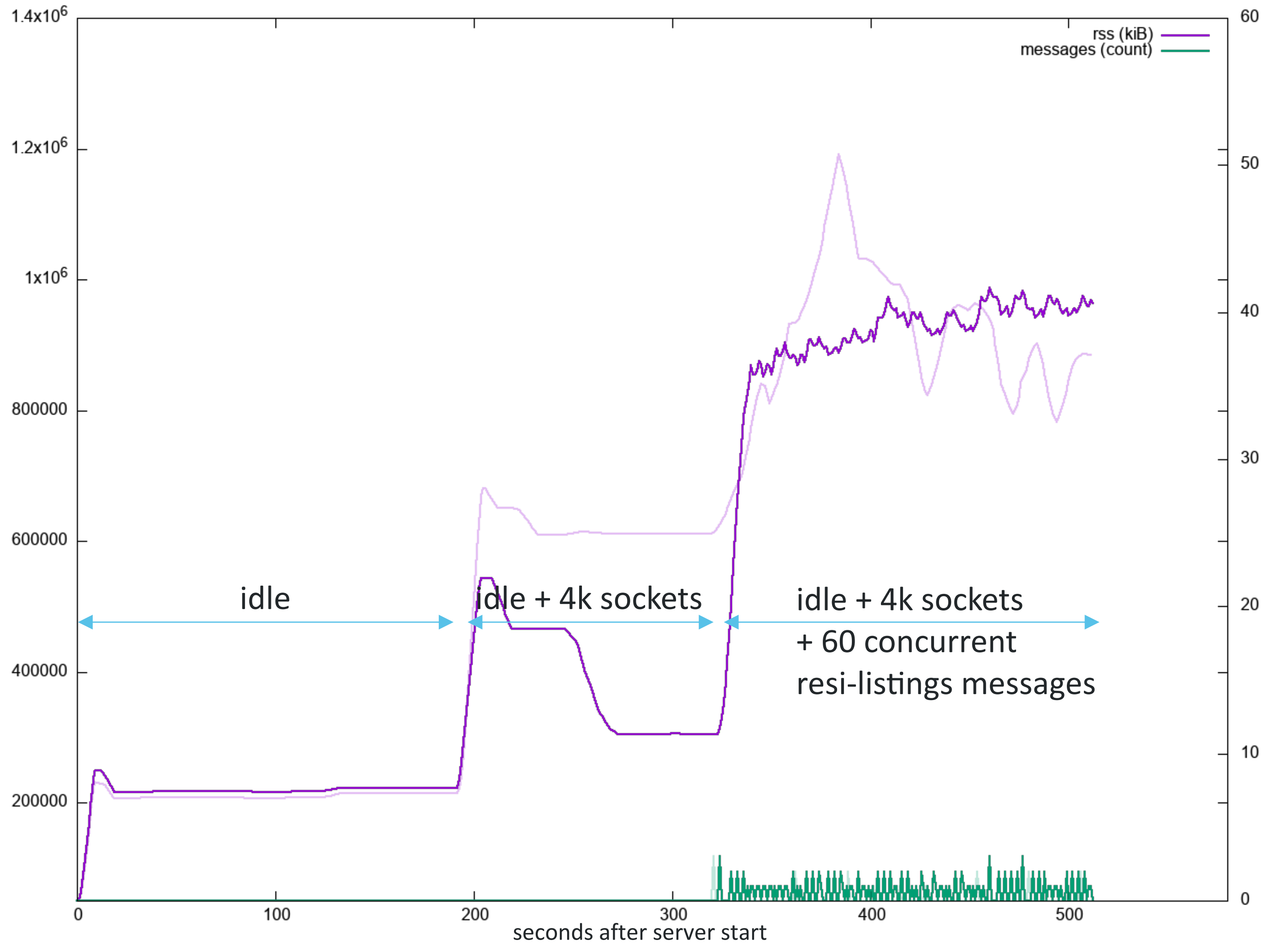
By default, the parser creates AST nodes that know the location in the source that they correspond to. This configuration flag disables that behavior for performance or testing.



```
diff --git a/node_modules/subscriptions-transport-ws/dist/server.js b/node_modules/subscriptions-transport-ws/dist/server.js
index 730c58506..dd27c859b 100644
--- a/node_modules/subscriptions-transport-ws/dist/server.js
+++ b/node_modules/subscriptions-transport-ws/dist/server.js
@@ -185,7 +185,7 @@ var SubscriptionServer = (function () {
                                 _this.sendError(connectionContext, opId, { message: error });
                                 throw new Error(error);
                             }
-                            var document = typeof baseParams.query !== 'string' ? baseParams.query : graphql_1.parse(baseParams.query);
+                            var document = typeof baseParams.query !== 'string' ? baseParams.query : graphql_1.parse(baseParams.query, {noLocation: true});
                             var executionPromise;
                             var validationErrors = graphql_1.validate(params.schema, document, _this.specifiedRules);
                             if (validationErrors.length > 0) {
```

No `Location` in Subscription AST



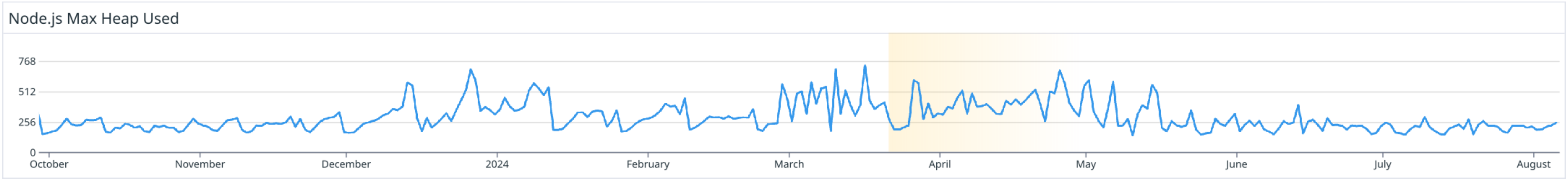


Message Counts (vs. baseline):

```
recv:     121 ▲ 6.562x
send:  484000 ▲ 7.642x
```

AUCTION.COM BEYOND THE BID.

Gate loaders with accessor + AST query cache



`dataloader` and primer objects are created unconditionally for each context, similarly to backend accessors. Why? We need a formalism for lazily building up objects of the form

```
{
  key1: functionCall1(),
  key2: functionCall2(),
  …
}
```

Enter

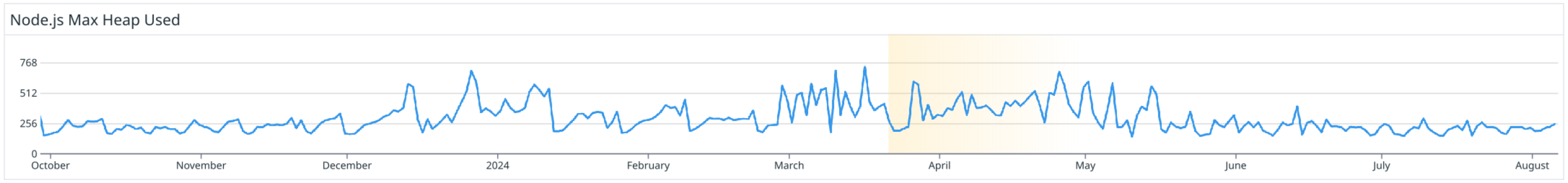
```
Proxy.lazy({
  key1: functionCall1(),
  key2: functionCall2(),
  …
})
```

Implemented as a babel plugin, it replaces the object literal with a proxy, transforming the object literal's contents into a ternary.

The generated code (roughly):

```
(() => {
  const sym =
    Symbol.for('propNotConstructed')
  const target = {key1: sym, key2: sym, …}
  const ternary = prop =>
    prop === key1 ? functionCall1() :
    prop === key2 ? functionCall2() :
    …
  return new Proxy(target, {
    get: (target, prop) =>
      target[prop] === sym
        ? (target[prop] = ternary(prop))
        : target[prop]
  })
})()
```

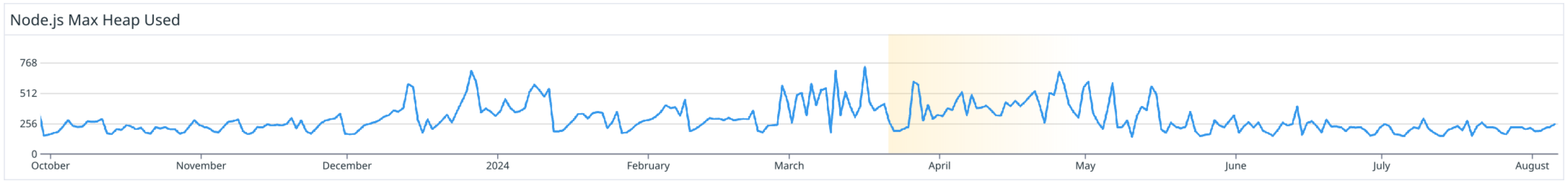
Gate loaders with accessor + AST query cache

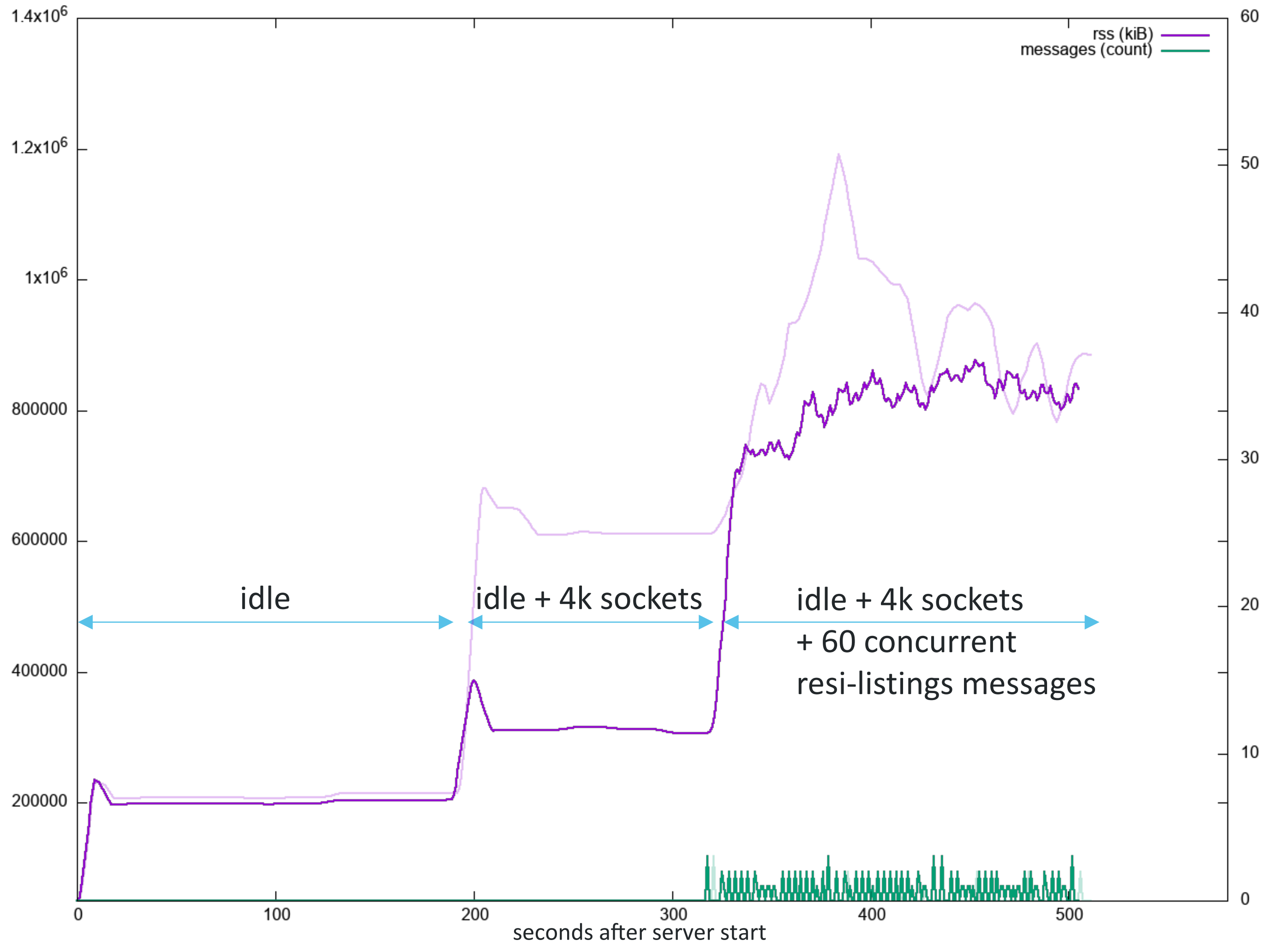


Monkey-patch graphql with AST query cache, since the subscriptions provider, unlike Apollo, doesn't have its own query cache:

- **`parse`**:
  - Takes SHA256 of incoming string
  - Looks up AST / compilation artifact from LRU cache and
    - either returns found item, or
    - creates AST, compiles, and attaches compilation artifact.
- **`execute`**:
  - Must be as lean as possible, because it happens for every message, for every connection.
  - Retrieve compilation artifact from LRU cache and,
    - either run compilation artifact if found, or
    - run regular query otherwise.

Gate loaders with accessor + AST query cache





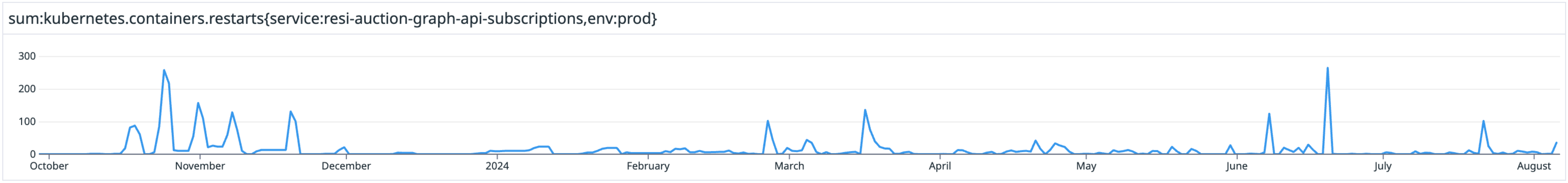
Message Counts (vs. baseline):

```
recv:     121 ▲ 6.562x
send:  484000 ▲ 7.642x
```

Future Directions

- Use heapDiff to identify more objects that can be removed
- Take another look at Promise objects
- Upgrade packages
- Create a realistic subscriptions load test

Insights



- Choose carefully what you attach to your context.
- Execution efficiency and memory usage efficiency are not entirely orthogonal.
    - A more efficient execution may result in fewer temporary objects.
- `node-clinic` for flame graphs is a great tool.
- Chrome Developer Tools heap snapshots (especially differential snapshots) – another great tool.

Insights

Questions?