@

How *Redis* plays a key role in the world's coolest case management solution

sebastian.dehne@systek.no

Skatteetaten

Systek

Code examples available

https://github.com/sebdehne/redis-at-skatt-talk

Agenda

- **Project SIRIUS**
- Locking with Redis
- Running processes with Redis
- Other Redis use-cases
- Conclusion / lessions learned

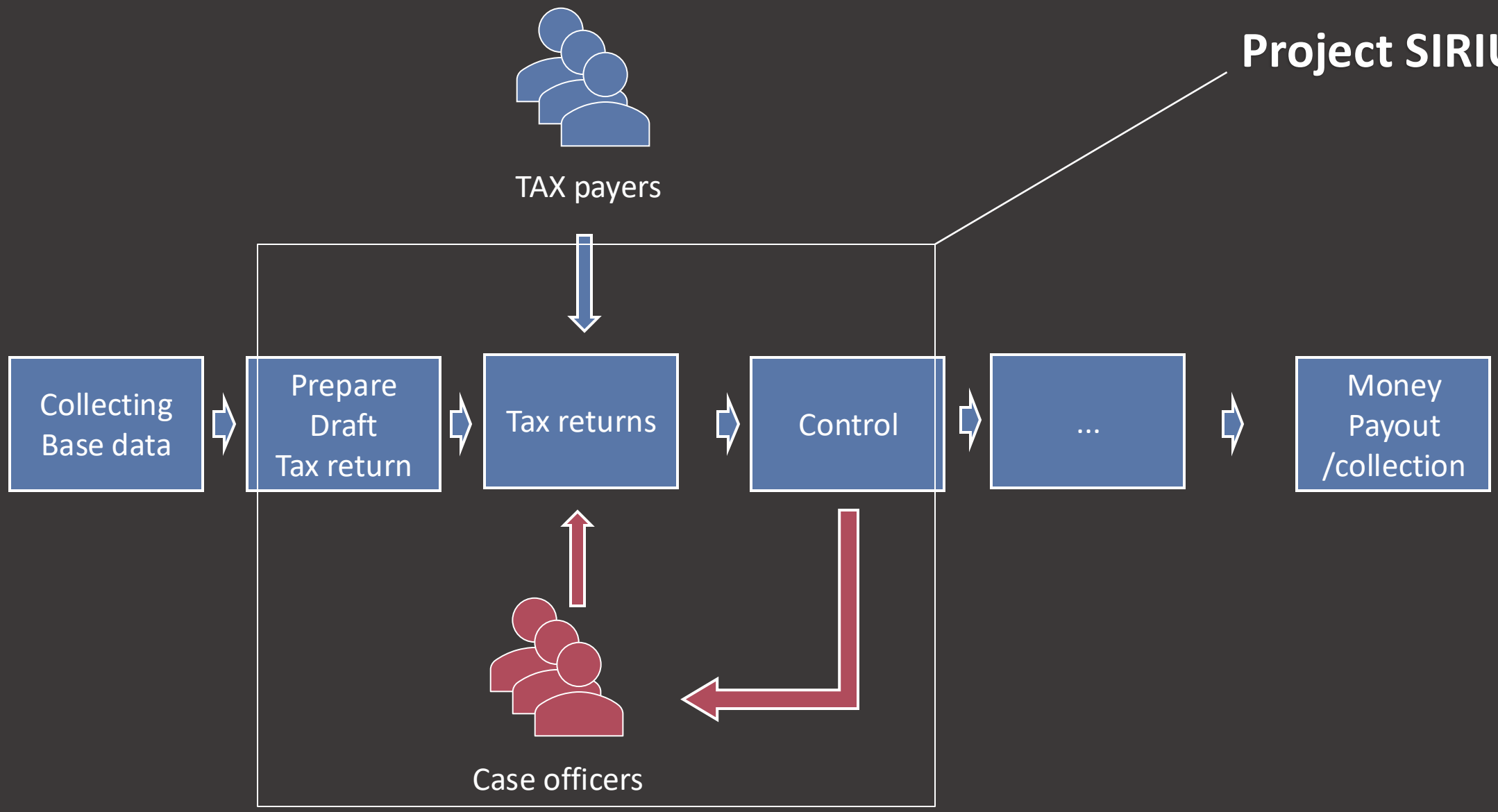
https://github.com/sebdehne/redis-at-skatt-talk

Skatteetaten

sebastian.dehne@systek.no

Systek

Project SIRIUS



sebastian.dehne@systek.no

Skatteetaten

Systek

Skatteetaten

Skattemeldingen 2023 - EVIG VÅGHALS

Skriv søkeord her

Din foreløpige skatt

Foreløpig beregnet skatt og avgift	441 699
Skatt du må betale (restskatt)	**441 699**

Du finner en detaljert utregning i oppsummeringen lenger ned på denne siden.

Har du lagt til eller endret opplysninger?

Send inn nye opplysninger | Gå ut av skattemeldingen

Status: Behandlet av Skatteetaten 31.07.2024 18:14.

Se hvem du skatter sammen med

Arbeid, trygd og pensjon

Lønn og tilsvarende ytelser

ertgrtg

Lønn	234 234

Åpne og endre

egerg

Lønn	234 324

Åpne og endre

Utenlandsk arbeidsgiver

Opptjeningsland	Bahamas
Lønn og annen utbetaling for arbeid i utlandet	50 000 BSD

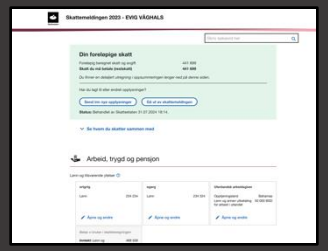
Åpne og endre

Beløp vi bruker i skatteberegningen

***Inntekt:** Lønn og* *468 558*

Project SIRIUS

6 million public TAX payers



“SME” (Tax return for the public)

sebastian.dehne@systek.no

Skatteetaten

Systek

Project SIRIUS

Skatteetaten

SMIA

Hjemmeside SMIA

Fraser | Fraseeditor | Maskinelle jobber | Oppgavetypedefinisjon | Saksadministrasjon og oppgavefordeling | Kontrolladministrasjon | Tilpass Smia | Hjelp

Mine oppgavefiltre

+ Nytt filter

Du har ingen filtre. Klikk på "+ Nytt filter" for å opprette et

Søk

Søk på navn, personnummer eller saksmappeidentifikator...

2023

Søk

Arbeidsliste

Tilgjengelige | Kvalitetssikring | Mine oppgaver (1) | Ferdigstilte

Inntektsår: Alle | Sakstype: Alle | Status: Alle

Inntektsår	F.nr/Org.nr	Gjelder	Arbeidsoppgave		Status	Svarfrist
2024	310596345	AKTVERDIG KOMPATIBEL TIGER AS	Fastsetting - Korrespondanse Utgående korrespondanse		Under arbeid	

Nyheter

Vis flere

Siste åpnede saker

AKTVERDIG KOMPATIBEL TIGER AS - 310596345
Oppgave - 2024 (15.08.2024 11:14:38)

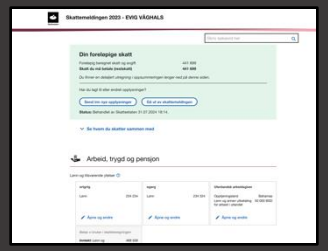
AKTVERDIG KOMPATIBEL TIGER AS - 310596345
Sak - 2025 (14.08.2024 11:00:44)

Tøm historikk

Project SIRIUS

6 million public TAX payers

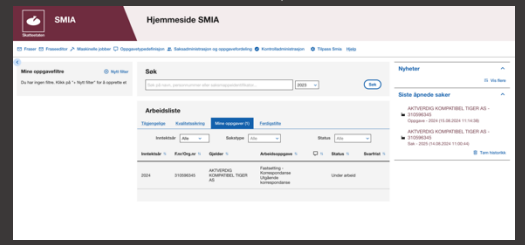
2500 internal Case officers



“SME” (Tax return for the public)

Content

Content



“SMIA” (Internal case Management)

Team Vulcan

Team IO

Team Laika

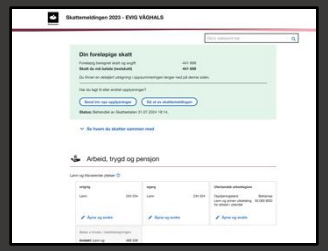
sebastian.dehne@systek.no

Skatteetaten

Systek

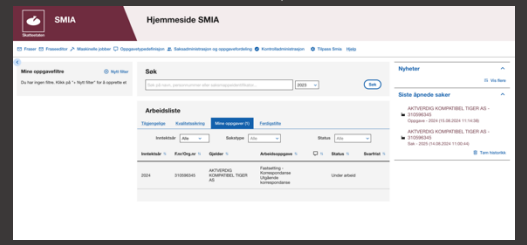
Project SIRIUS Architecture

6 million public TAX payers



“SME”
(Tax return for the public)

2500 internal Case officers



“SMIA”
(Internal case Management)

Document store (key/value)

Journal store for Meta data

Tickets

sebastian.dehne@systek.no

Skatteetaten

Systek

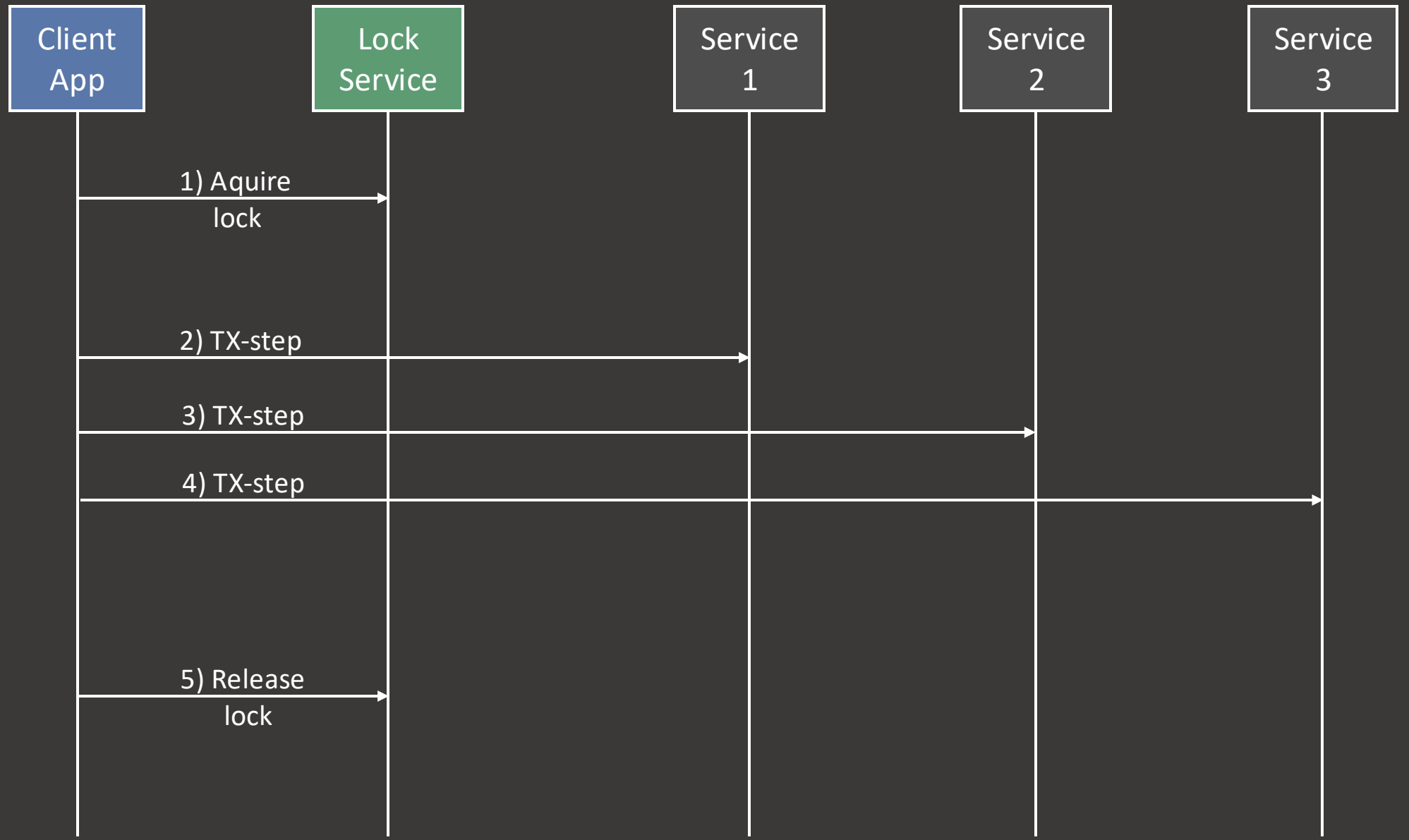
A typical transaction…

runLocked(customerId) {

1. Read current version of TAX return
2. Update and write new Tax return to document store
3. Update Journal store with new meta data
4. Send out letter
5. Update Journal store with sent letter meta data
6. Update (/close) ticket
7. Updated search indexes

}

sebastian.dehne@systek.no



sebastian.dehne@systek.no

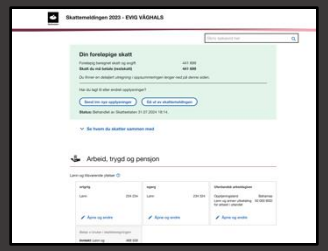
Skatteetaten

Systek

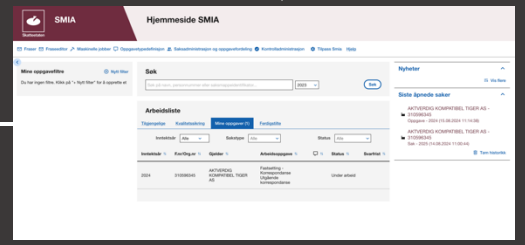
SIRIUS Architecture

6 million public TAX payers

2500 internal Case officers



"SME" (Tax return for the public)



"SMIA" (Internal case Management)

Lock Service

Backend

Document store (key/value)

Journal store for Meta data

Tickets

sebastian.dehne@systek.no

Skatteetaten

Systek

Possible solutions for “Lock Service”

- Postgres Advisory Locks
- Use Redis as a lock-service

sebastian.dehne@systek.no

Agenda

- Project SIRIUS
- **Locking with Redis**
- Running processes with Redis
- Other Redis use-cases
- Conclusion / lessions learned

https://github.com/sebdehne/redis-at-skatt-talk

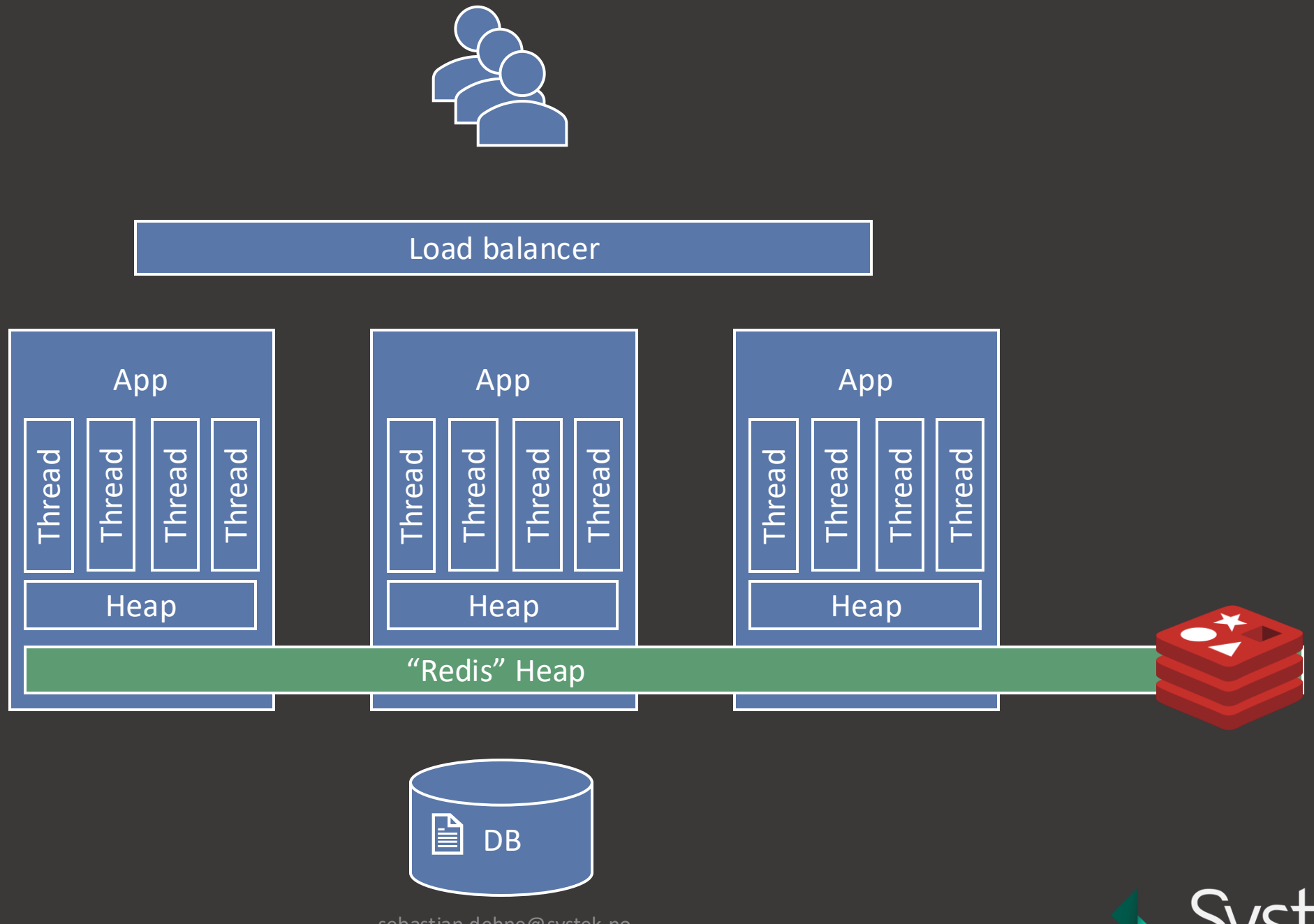
What is Redis

- In-memory key/value database
- Easy to use / reason about
- Very lightweight
  - Based on single-thread event loop
- High performant
- No security (out of the box)
- Redis is no longer open source
- Linux foundation created a fork: Valkey

sebastian.dehne@systek.no

Demo

sebastian.dehne@systek.no



sebastian.dehne@systek.no

Skatteetaten

Systek

Redis features

- Data store:
  - Strings
  - Lists
  - Hashes
  - Sets
  - Sorted sets
- Publish/Subscribe/Notify
- Transactions
- Scripting

sebastian.dehne@systek.no

Our Redis setup

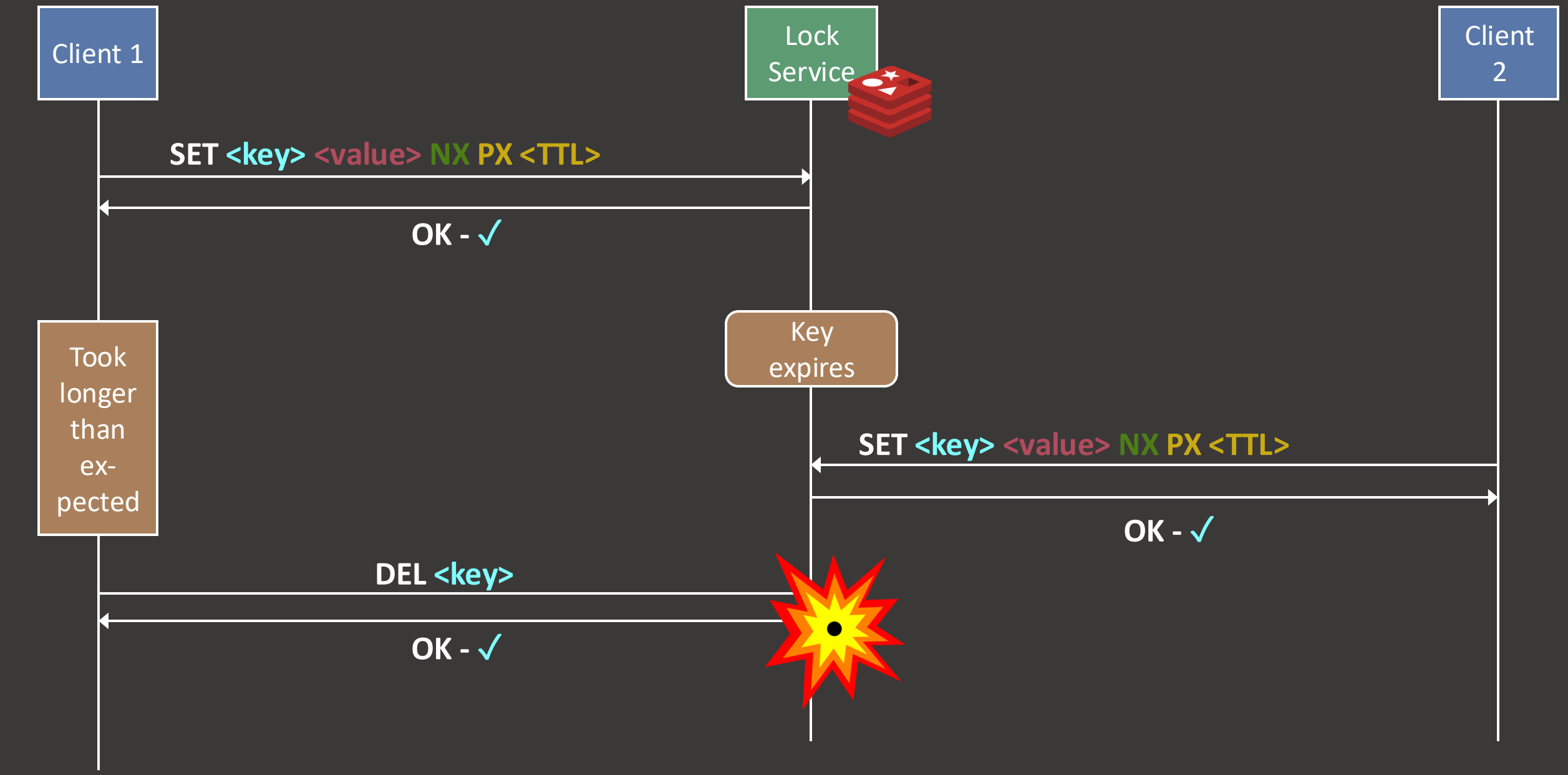
- Single instance – no cluster (KISS)
- Disabled disk storage – in memory only (use DB for persistence)
- Suitable for short lived state
- Which can be lost at any time
- Dataset small enough to fit into RAM
- Use TTL / expire
- Name spacing to avoid conflicts

sebastian.dehne@systek.no

Let’s design a basic lock in Redis (“*RedLock*”)

```
SET <key> <value> NX PX <TTL>     // lock

DEL <key>                         // unlock
```


sebastian.dehne@systek.no



sebastian.dehne@systek.no

Skatteetaten

Systek

Let's design a basic lock in Redis ("*RedLock*")

SET <key> <random_value> NX PX <TTL>

DEL <key> **// only if value == <random_value>**

```
if redis.call('get', KEYS[1]) == ARGV[1] then
    return redis.call('del', KEYS[1])
else
    return 0
end
```

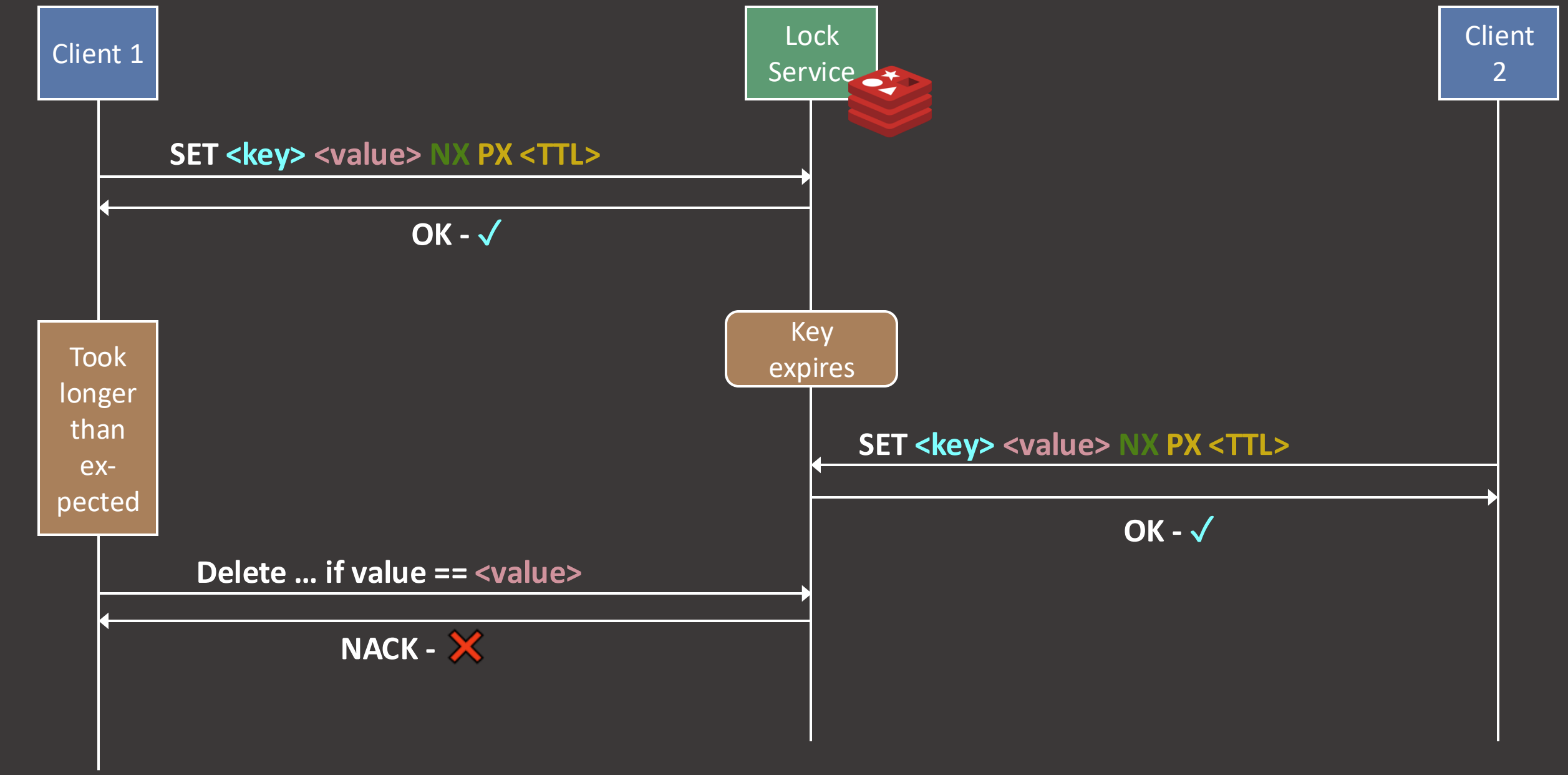

sebastian.dehne@systek.no

Let’s design a basic lock in Redis (“*RedLock*”)

SET <key> <random_value> NX PX <TTL>

EVAL "*if redis.call('get',KEYS[1]) == ARGV[1] then return redis.call('del',KEYS[1]) else return 0 end*" **1 <key> <random_value>**

sebastian.dehne@systek.no



sebastian.dehne@systek.no

Skatteetaten

Systek

```kotlin
    private fun namespaceKey(lock: String) = "${this::class.java.name}-$lock"

    private fun lock(lock: String, expiresSeconds: Long = 30): String? {
        val key = namespaceKey(lock)

        val randomValue = UUID.randomUUID().toString()
        return if (jedis.set(
                key,
                randomValue,
                SetParams()
                    .nx()
                    .ex(expiresSeconds)
            ) == null
        ) {
            null
        } else {
            randomValue
        }
    }
```

sebastian.dehne@systek.no

Skatteetaten

Systek

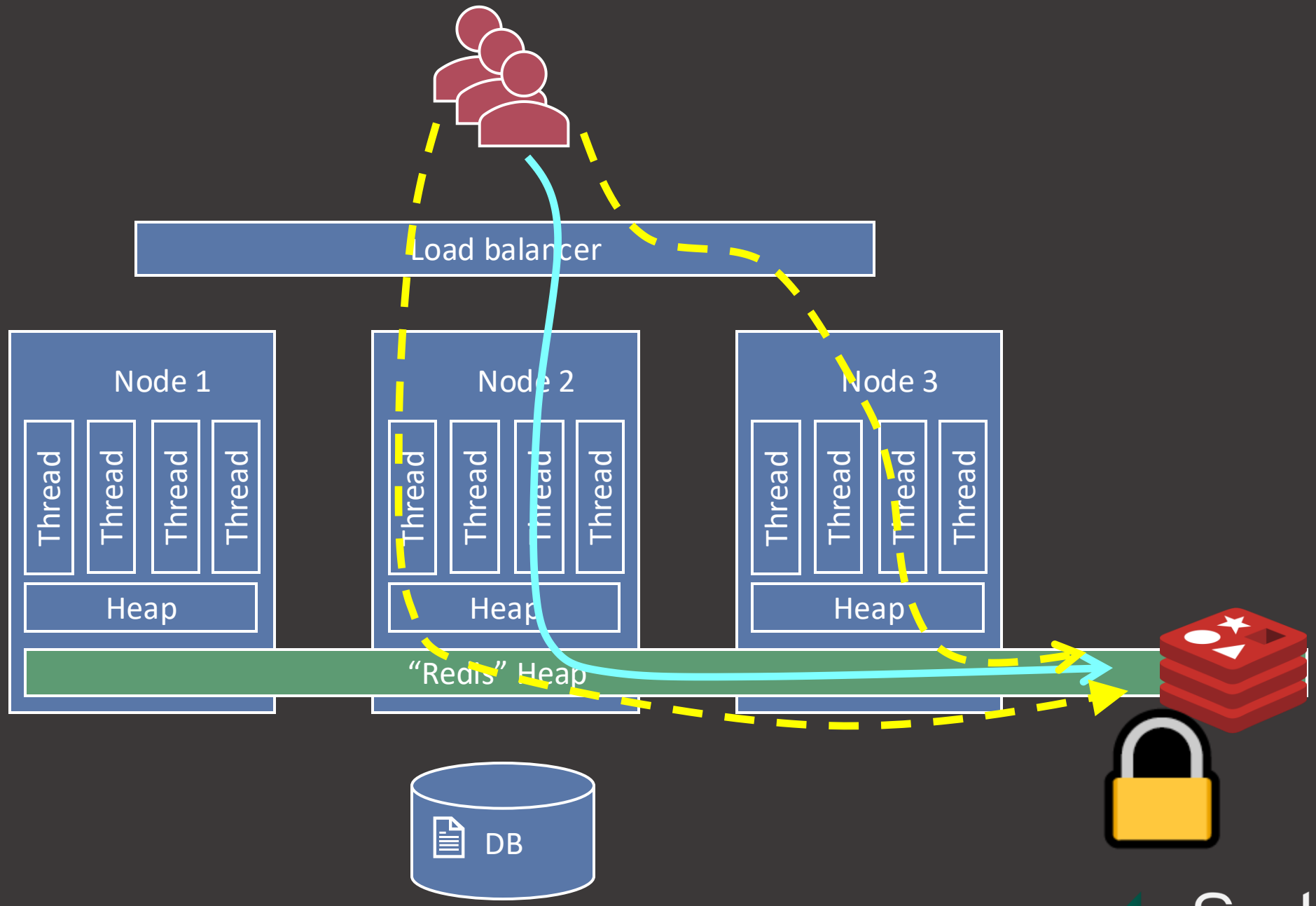
```
29    private fun unlock(lock: String, randomValue: String) {
30        val key = namespaceKey(lock)
31
32        jedis.eval(
33            """
34            if redis.call('get', KEYS[1]) == ARGV[1] then
35                return redis.call('del', KEYS[1])
36            else
37                return 0
38            end
39            """.trimIndent(),
40            listOf(key),
41            listOf(randomValue)
42        )
43    }
```

```kotlin
    fun tryRunLocked(
        lock: String,
        expiresSeconds: Long = 30,
        fn: () -> Unit
    ): Boolean {
        val randomValue = lock(lock, expiresSeconds) ?: return false
        return try {
            fn()
            true
        } finally {
            unlock(lock, randomValue)
        }
    }
```

Skatteetaten

sebastian.dehne@systek.no

Systek



sebastian.dehne@systek.no

Skatteetaten

Systek

Redis transactions

- All Redis commands are atomic
  - Redis is single threaded (event-loop)

- Use Redis transactions for **read-modify-write**

sebastian.dehne@systek.no

Redis transactions

1. WATCH <key>
2. GET <key>
3. // eval value in code
4. MULTI
5. SET <key> <new value>
6. EXEC

Demo

sebastian.dehne@systek.no

Better locks

- Want to capture more information:
  - Track who owns the lock at a given moment in time
  - See who had it last
- Instead of storing a <random_value>, we store an object “Lock”:

```
90    data class Lock(
91        val id: String,
92        val ownerHost: String,
93        val ownerThread: String,
94        val validUntil: Long,
95        val updatedAt: Instant,
96    )
97
98    fun Lock?.isAvailable(time: Long) = if (this == null) true else validUntil < time
```

```
49      private fun readModifyWrite(lock: String, fn: (existing: Lock?) -> Lock?): Boolean {
50          val key = namespaceKey(lock)
51
52          for (i in 0 ≤ .. ≤ 100) {
53              jedis.watch(key)
54              val existing = jedis.get(key)?.let {
55                  objectMapper.readValue<Lock>(it)
56              }
57
58              val updated = fn(existing) ?: return false
59
60              val ok = jedis.multi().let { tx ->
61                  tx.set(
62                      key,
63                      objectMapper.writeValueAsString(updated)
64                  )
65                  tx.exec().size == 1
66              }
67
68              if (ok) return true
69          }
70          error("Too many retries")
71      }
```

sebastian.dehne@systek.no

Skatteetaten

Systek

```kotlin
15      private fun lock(lock: String, expiresSeconds: Long = 30): String? {
16          val now = Instant.now()
17          var lockId: String? = null
18
19          readModifyWrite(lock) { existingLock ->
20              if (!existingLock.isAvailable(now.toEpochMilli())) {
21                  return@readModifyWrite null
22              }
23
24              Lock(
25                  id = UUID.randomUUID().toString(),
26                  ownerHost = System.getenv( name: "POD_NAME") ?: "unknown",
27                  ownerThread = Thread.currentThread().name,
28                  validUntil = now.toEpochMilli() + (expiresSeconds * 1000),
29                  updatedAt = now
30              ).apply { lockId = id }
31          }
32
33          return lockId
34      }
```

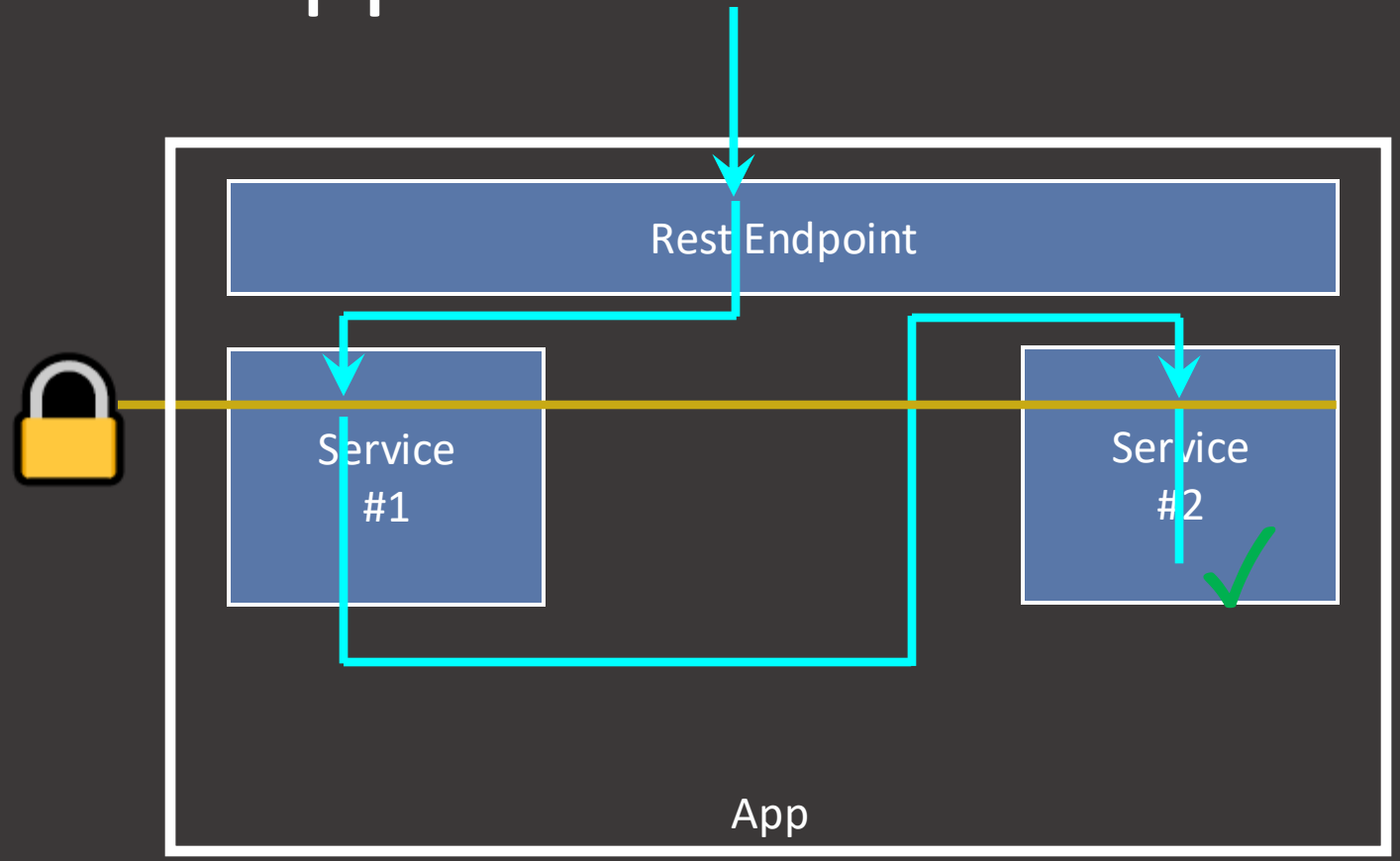
sebastian.dehne@systek.no

Skatteetaten

Systek

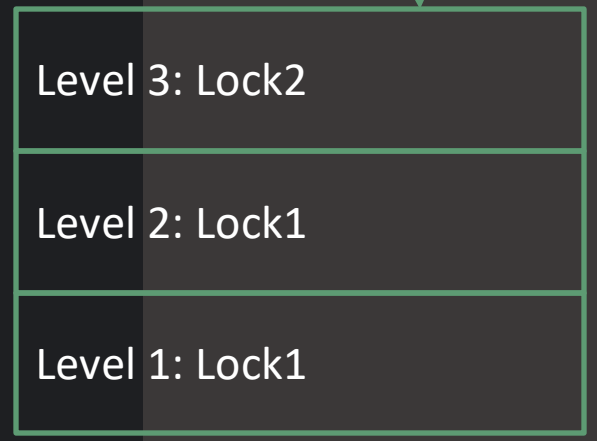
```
36      private fun unlock(lock: String, id: String) {
37          readModifyWrite(lock) { existingLock ->
38              if (existingLock?.id != id) {
39                  null
40              } else {
41                  existingLock.copy(
42                      validUntil = 0,
43                      updatedAt = Instant.now()
44                  )
45              }
46          }
47      }
```

Reentrant support



sebastian.dehne@systek.no

```
15      private val reentrant: ThreadLocal<LinkedList<String>> = ThreadLocal()
16
17      fun tryRunLocked(lock: String, expiresSeconds: Long = 30, fn: () -> Unit): Boolean {
18          var lockId: String? = null
19
20          if (reentrant.get() == null) { reentrant.set(LinkedList()) } // first entry
21          if (lock !in reentrant.get()!!) { // new, unknown lock
22              lockId = lock(lock, expiresSeconds) ?: return false
23          }
24          reentrant.get()!!.addLast(lock)
25
26          return try {
27              fn()
28              true
29          } finally {
30              if (lockId != null) {
31                  unlock(lock, lockId)
32              }
33              reentrant.get()!!.removeLast()
34              if (reentrant.get()!!.isEmpty()) {
35                  reentrant.set(null) // last exit, cleanup
36              }
37          }
38      }
```



sebastian.dehne@systek.no

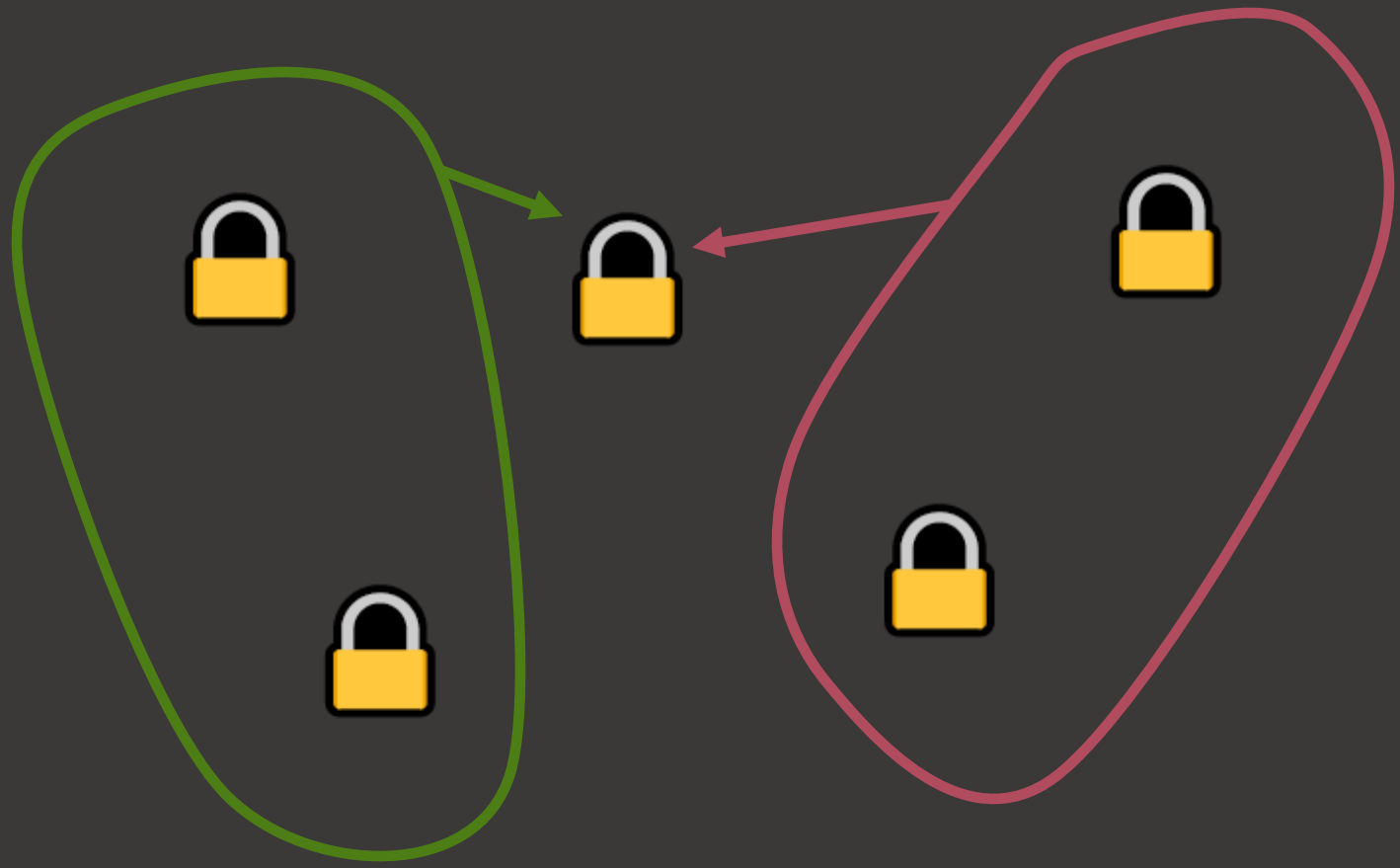
Skatteetaten

Systek

Acquiring multiple locks

sebastian.dehne@systek.no

Acquiring multiple locks & Reentrant



sebastian.dehne@systek.no

```
77      private fun readModifyWrite(locks: Set<String>, fn: (existing: Map<String, Lock?>) -> Lock?): Boolean {
78          val keys = locks.map { namespaceKey(it) }
79
80          for (i in 0 ≤ .. ≤ 100) {
81              jedis.watch(*keys.toTypedArray())
82
83              val existing = keys.associateWith { l ->
84                  jedis.get(l)?.let { objectMapper.readValue<Lock>(it) }
85              }
86
87              val updated = fn(existing) ?: return false
88
89              val ok = jedis.multi().let { tx ->
90                  keys.forEach { l -> tx.set(l, objectMapper.writeValueAsString(updated)) }
91
92                  tx.exec().size == keys.size
93              }
94
95              if (ok) return true
96          }
97          error("Too many retries")
98      }
```

```kotlin
    private fun lock(locks: Set<String>, expiresSeconds: Long = 30): String? {
        val now = Instant.now()
        var lockId: String? = null

        readModifyWrite(locks) { existing ->
            if (!existing.values.all { it.isAvailable(now.toEpochMilli()) }) {
                return@readModifyWrite null
            }

            Lock(
                id = UUID.randomUUID().toString(),
                ownerHost = System.getenv( name: "POD_NAME") ?: "unknown",
                ownerThread = Thread.currentThread().name,
                validUntil = now.toEpochMilli() + (expiresSeconds * 1000),
                updatedAt = now
            ).apply { lockId = id }
        }

        return lockId
    }
```

```
63      private fun unlock(locks: Set<String>, id: String) {
64          readModifyWrite(locks) { existing ->
65
66              if (existing.values.all { it?.id == id }) {
67                  existing.values.firstOrNull()?.copy(
68                      validUntil = 0,
69                      updatedAt = Instant.now()
70                  )
71              } else {
72                  null
73              }
74          }
75      }
```

```kotlin
15      private val reentrant: ThreadLocal<LinkedList<Set<String>>> = ThreadLocal()
16
17      fun tryRunLocked(locks: Set<String>, expiresSeconds: Long = 30, fn: () -> Unit): Boolean {
18          var lockId: String? = null
19
20          if (reentrant.get() == null) { reentrant.set(LinkedList()) }
21          val newLocks = locks - (reentrant.get()!!.flatten().toSet())
22
23          if (newLocks.isNotEmpty()) { // first time entry
24              lockId = lock(newLocks, expiresSeconds) ?: return false
25          }
26          reentrant.get()!!.add(newLocks)
27
28          return try {
29              fn()
30              true
31          } finally {
32              if (lockId != null) {
33                  unlock(newLocks, lockId)
34              }
35              reentrant.get()!!.removeLast()
36              if (reentrant.get()!!.isEmpty()) {
37                  reentrant.set(null) // last exit, cleanup
38              }
39          }
40      }
```

sebastian.dehne@systek.no

Skatteetaten

Systek

Lock API – different types

lock():

- blocks until available

tryLock(): Boolean

- returns immediately, returning true if acquired; false if unavailable

tryLock(t: Duration): Boolean

- Blocks until success or timeout

Redis Publish/Subscribe/Notify

- Demo

sebastian.dehne@systek.no

Redis Publish/Subscribe/Notify

- At-most-once - best effort
- Listening requires one dedicated TCP connection
  - we use a single subscription with sub-channels on top
  - distribute incoming messages internally
  - robustness - auto restart of this listener (in case of I/O errors)

sebastian.dehne@systek.no

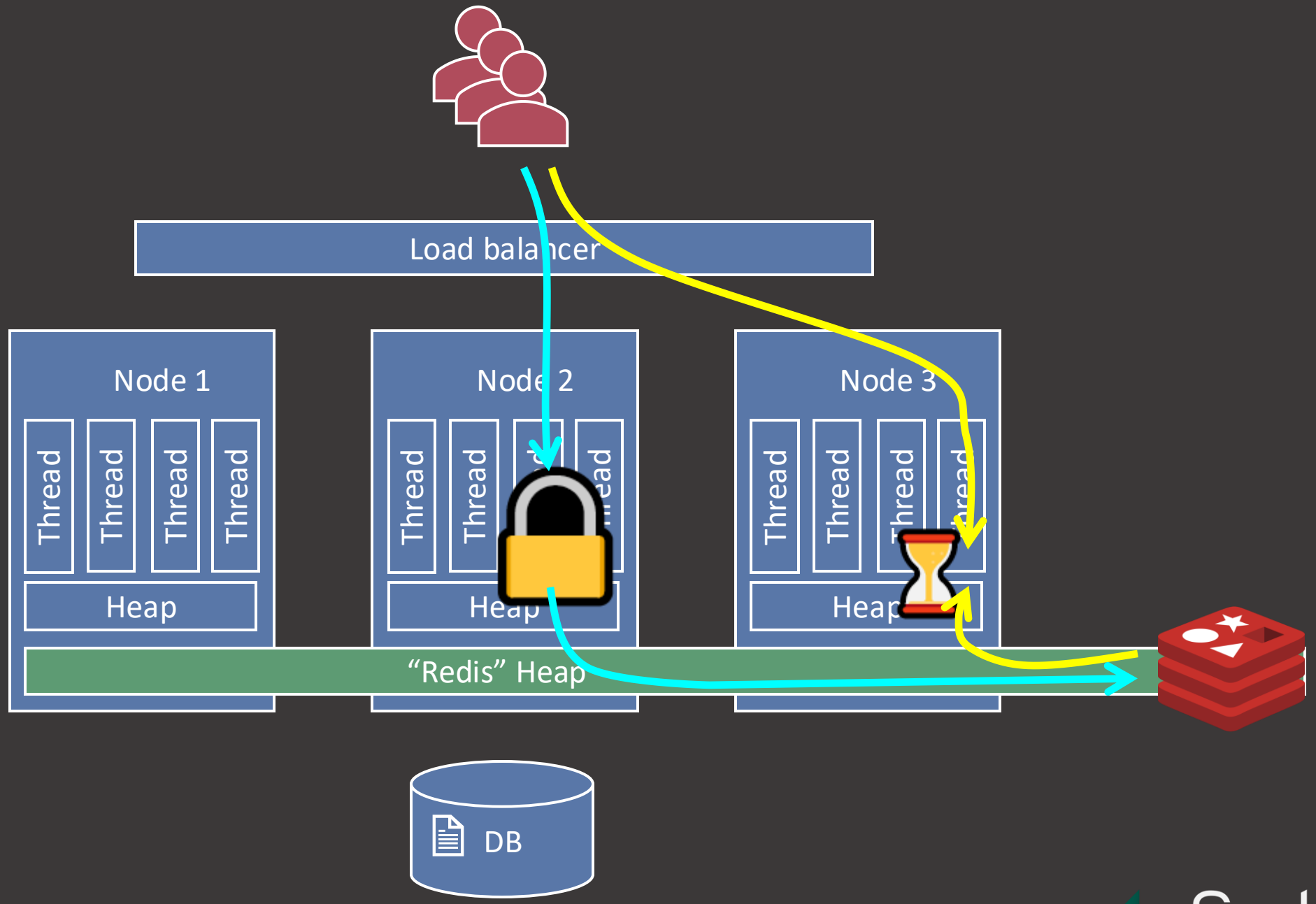
```kotlin
fun tryRunLocked(
    locks: Set<String>,
    timeout: Duration,
    lockTTL: Long = 30,
    fn: () -> Unit
): Boolean {
    if (tryRunLocked(locks, lockTTL, fn)) return true

    val deadLine = System.nanoTime() + timeout.toNanos()
    val sub = groupChatService.subscribe(channelIds = locks.map { namespaceKey(it) }.toSet())
    try {

        while (System.nanoTime() < deadLine) {
            if (tryRunLocked(locks, lockTTL, fn)) return true

            sub.poll(Duration.ofSeconds( seconds: 10))
        }
        return false

    } finally {
        sub.unsubscribe()
    }
}
```

sebastian.dehne@systek.no

Skatteetaten

Systek

```
94      private fun unlock(locks: Set<String>, id: String) {
95          var unlocked = false
96          readModifyWrite(locks) { existing ->
97
98              if (existing.values.all { it?.id == id }) {
99                  unlocked = true
100                 existing.values.first()!!.copy(
101                     validUntil = 0,
102                     updatedAt = Instant.now()
103                 )
104             } else {
105                 null
106             }
107         }
108         if (unlocked) {
109             groupChatService.notify(channelIds = locks.map { namespaceKey(it) }.toSet(), msg = "unlocked")
110         }
111     }
```



sebastian.dehne@systek.no

Skatteetaten

Systek

Automatic extending expire time

- No control over how long a lock is being held
- Need a way to extend the expire time, just in case

sebastian.dehne@systek.no

```kotlin
109     private fun extendTTL(lockId: String, locks: Set<String>, expiresSeconds: Long): Boolean =
110         readModifyWrite(locks) { existing ->
111             val now = Instant.now()
112             if (!existing.values.all { it.isAvailable(now.toEpochMilli()) && it?.id == lockId }) {
113                 null
114             } else {
115                 existing.values.first()!!.copy(
116                     updatedAt = now,
117                     validUntil = now.toEpochMilli() + (expiresSeconds + 1000)
118                 )
119             }
120         }
```

```
47      fun tryRunLocked(
48          locks: Set<String>,
49          expiresSeconds: Long = 30,
50          fn: (isStillHoldingLock: () -> Boolean) -> Unit
51      ): Boolean {
52          var lockId: String? = null
53
54          // ...
55
56          val isHoldingLock = AtomicBoolean( initialValue: true)
57          val autoExtendingTask = lockId?.let {
58              timer.scheduleWithFixedDelay({
59                  executorService.submit {
60                      if (isHoldingLock.get()) {
61                          isHoldingLock.set(extendTTL(lockId, locks, expiresSeconds))
62                      }
63                  }
64              }, initialDelay: expiresSeconds / 2, delay: expiresSeconds / 2, TimeUnit.SECONDS)
65          }
66
67          return try {
68              fn { isHoldingLock.get() }
69              true
70          } finally {
71              autoExtendingTask?.cancel( mayInterruptIfRunning: false)
72              // ..
73          }
74      }
```

```
16  class LockService(...) {
22
23      fun tryRunLocked(
24          locks: Set<String>,
25          timeout: Duration,
26          lockTTL: Long = 30,
27          fn: (isStillHoldingLock: () -> Boolean) -> Unit
28      ): Boolean {...}
46
47      fun tryRunLocked(
48          locks: Set<String>,
49          expiresSeconds: Long = 30,
50          fn: (isStillHoldingLock: () -> Boolean) -> Unit
51      ): Boolean {...}
88
89      private fun tryRunLocked(locks: Set<String>, expiresSeconds: Long = 30): String? {...}
109     private fun extendTTL(lockId: String, locks: Set<String>, expiresSeconds: Long): Boolean ={...}
121     private fun unlock(locks: Set<String>, id: String) {...}
139
140     private fun readModifyWrite(locks: Set<String>, fn: (existing: Map<String, Lock?>) -> Lock?): Boolean {...}
162
163     private fun namespaceKey(lock: String) = "${this::class.java.name}-$lock"
164     private val reentrant: ThreadLocal<LinkedList<Set<String>>> = ThreadLocal()
165     private val timer = Executors.newSingleThreadScheduledExecutor()
166
167 }
```

sebastian.dehne@systek.no

Skatteetaten

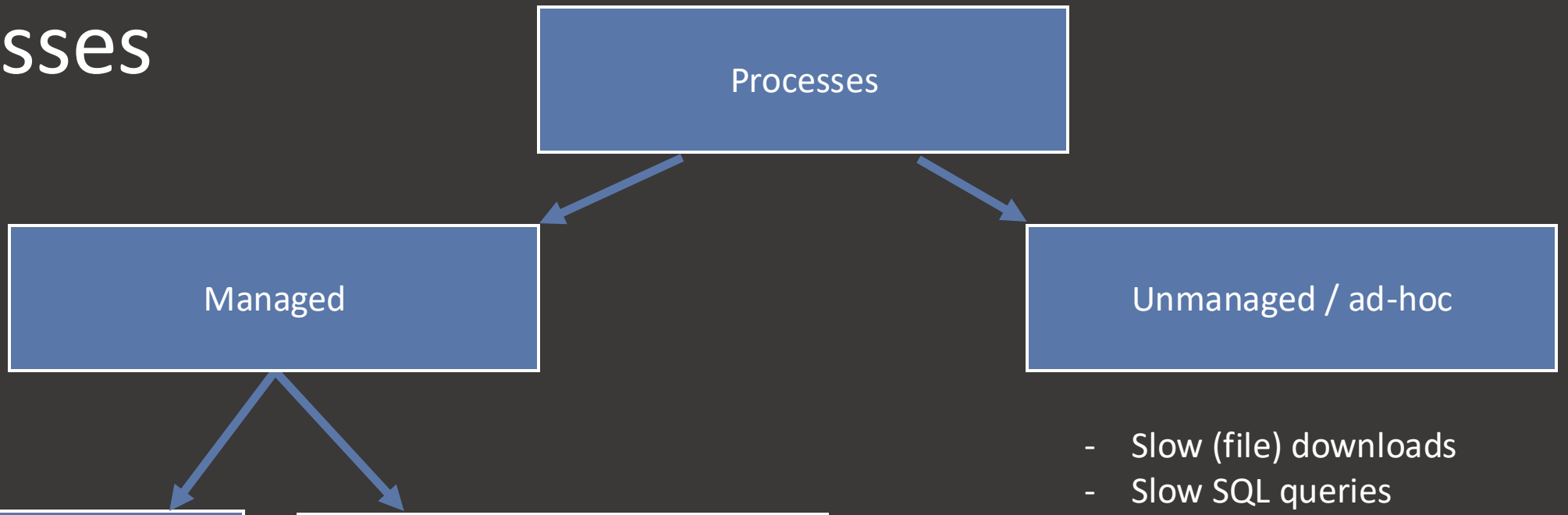
Systek

Agenda

- Project SIRIUS
- Locking with Redis
- **Running processes with Redis**
- Other Redis use-cases
- Conclusion / lessions learned

https://github.com/sebdehne/redis-at-skatt-talk

Processes



- Slow (file) downloads
- Slow SQL queries
- Typically something triggered by a user

- Read bulkdata from database
- DB-maintenance
- DB-migrations
- Read Atom-feeds
- File processing
- …

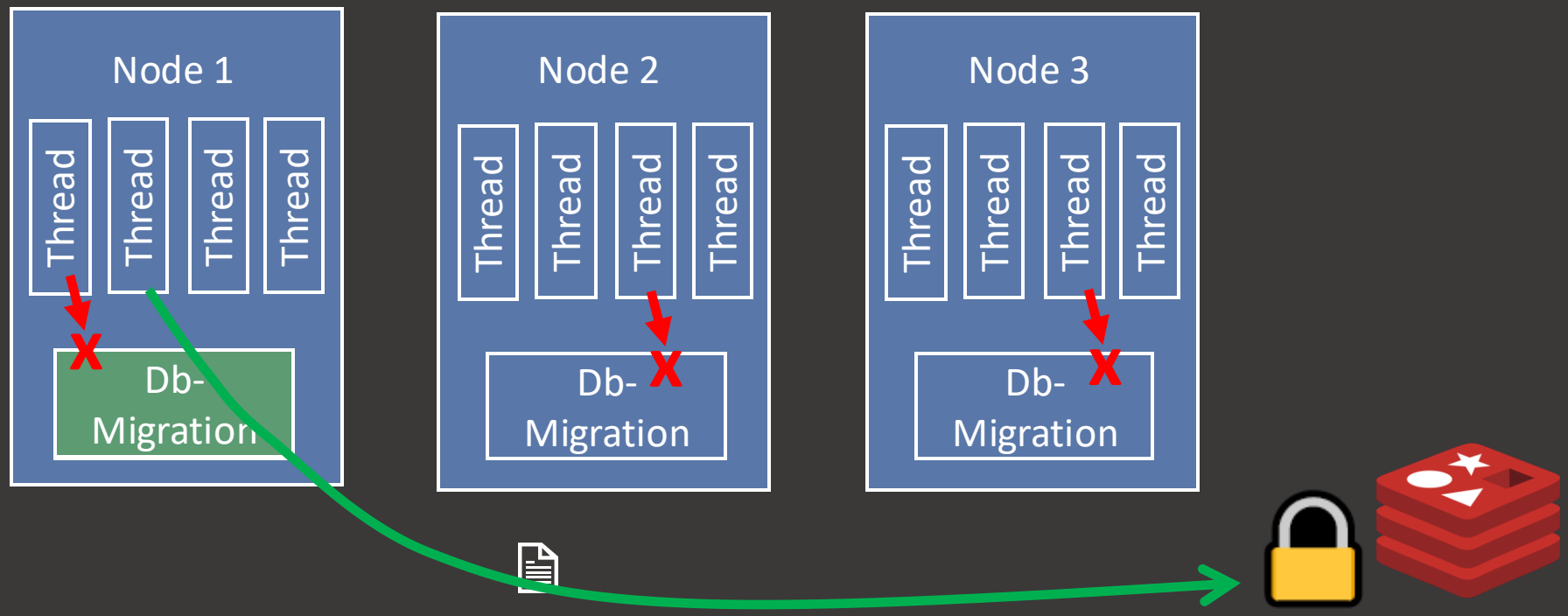
- Kafka consumers
- Task processing
- …

sebastian.dehne@systek.no

Skatteetaten

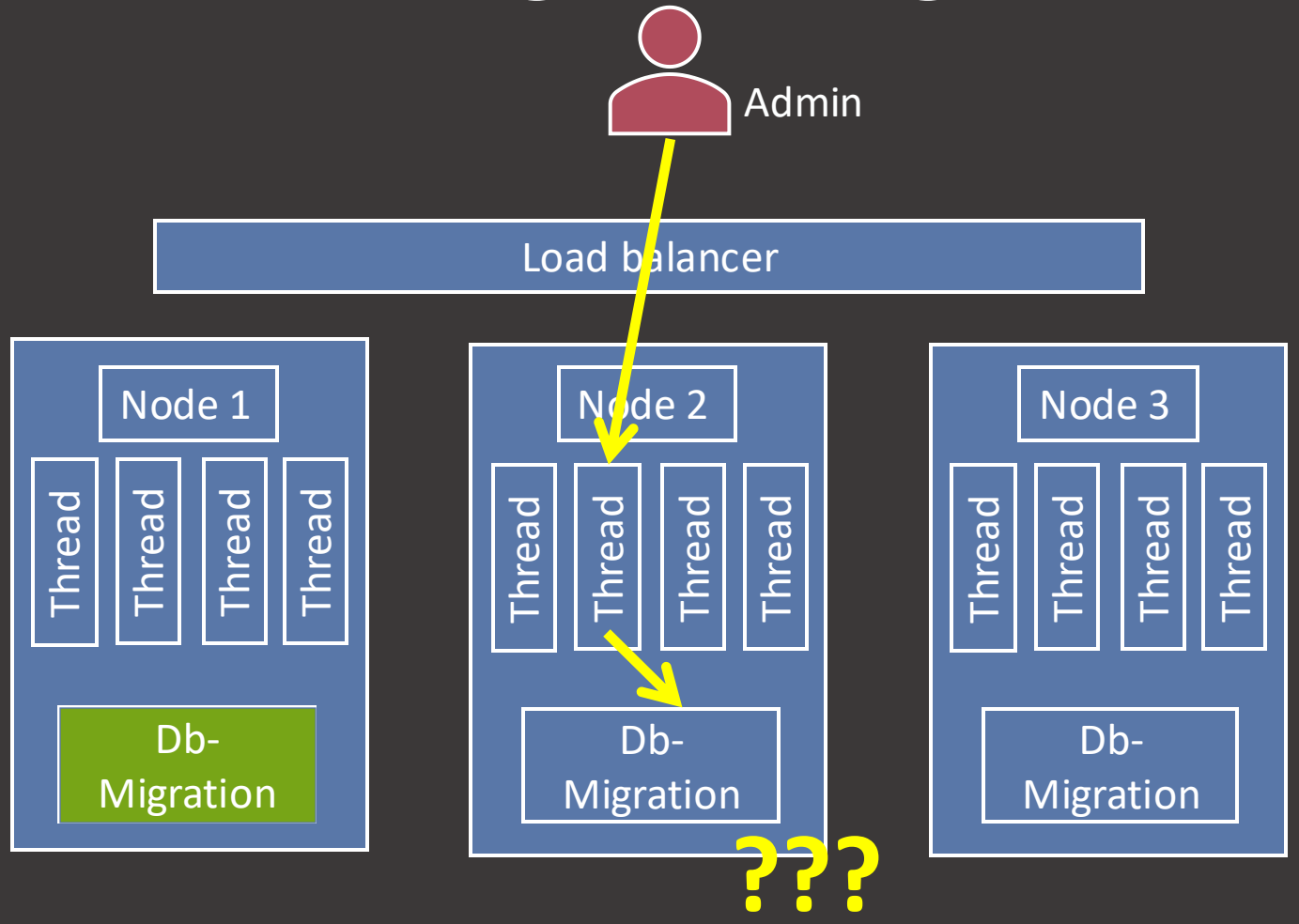
Systek

Processes / Managed / Single



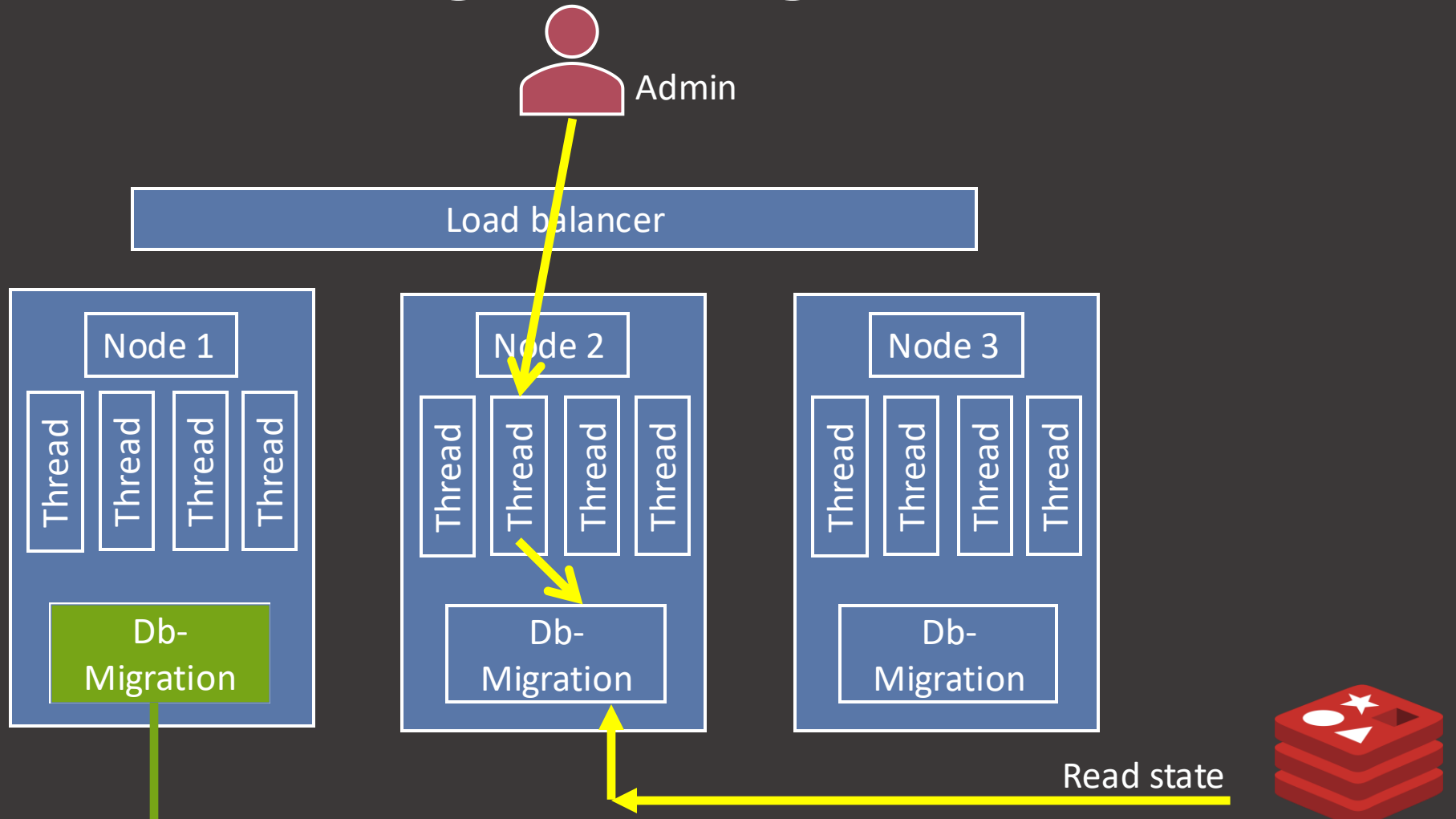
sebastian.dehne@systek.no

Processes / Managed / Single



sebastian.dehne@systek.no

Processes / Managed / Single

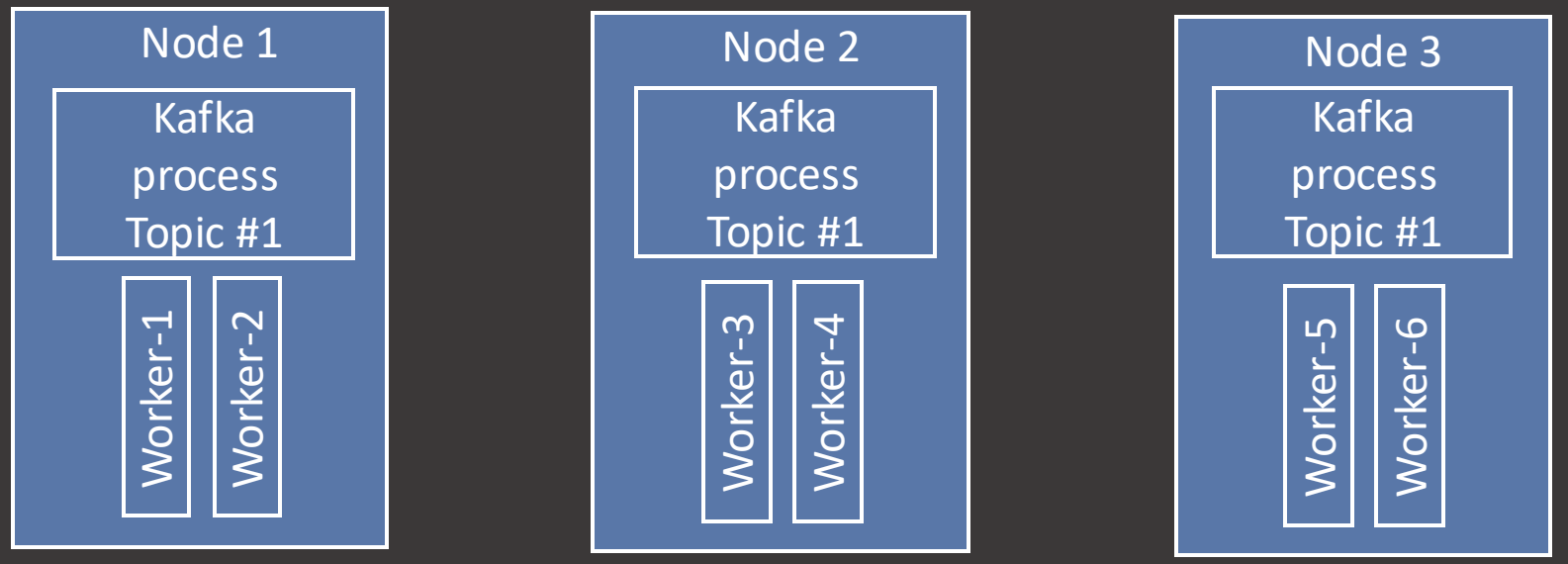


sebastian.dehne@systek.no

Skatteetaten

Systek

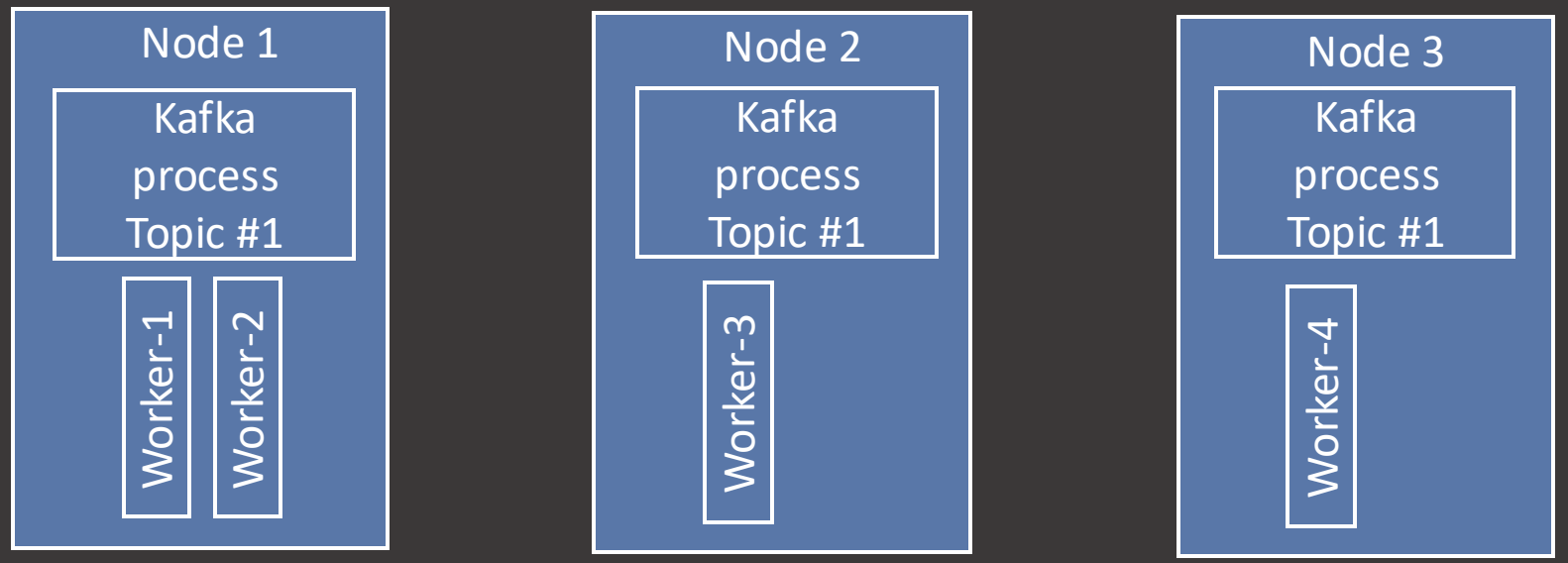
Processes / Managed / Concurrent



Admin

```
Config {
  workers: 6
}
```

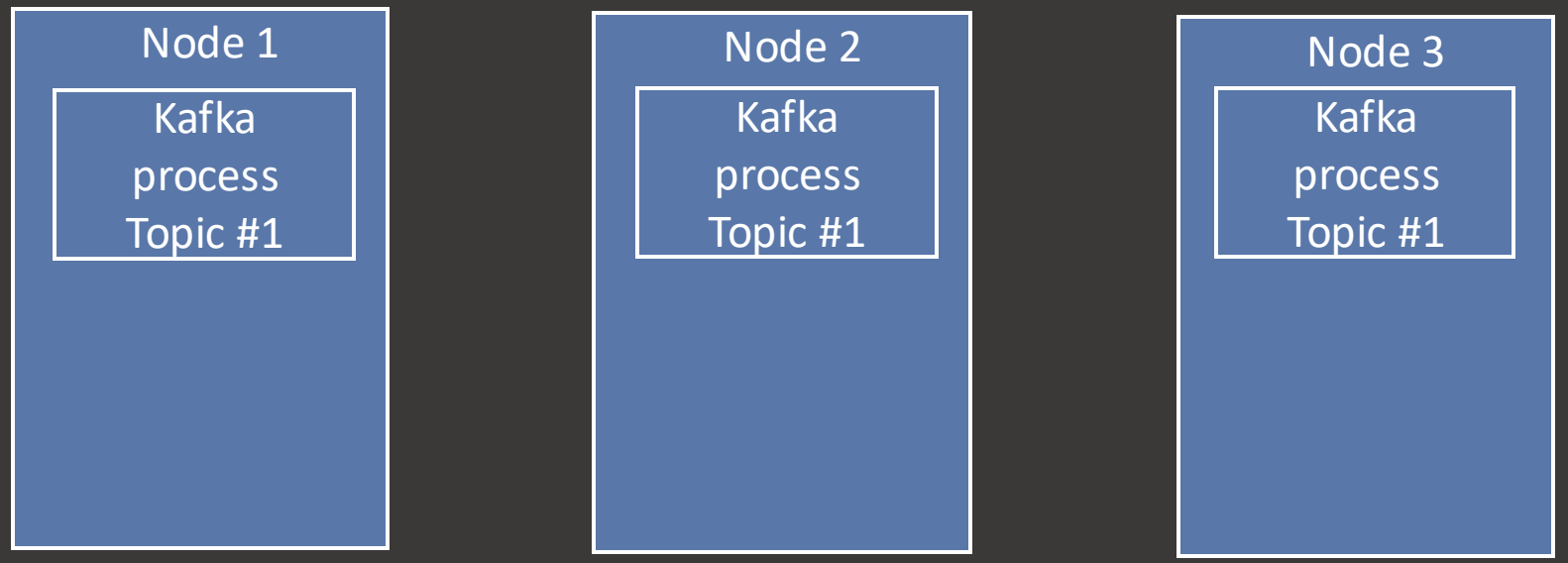
Processes / Managed / Concurrent




```
Config {
  workers: 4
}
```


sebastian.dehne@systek.no

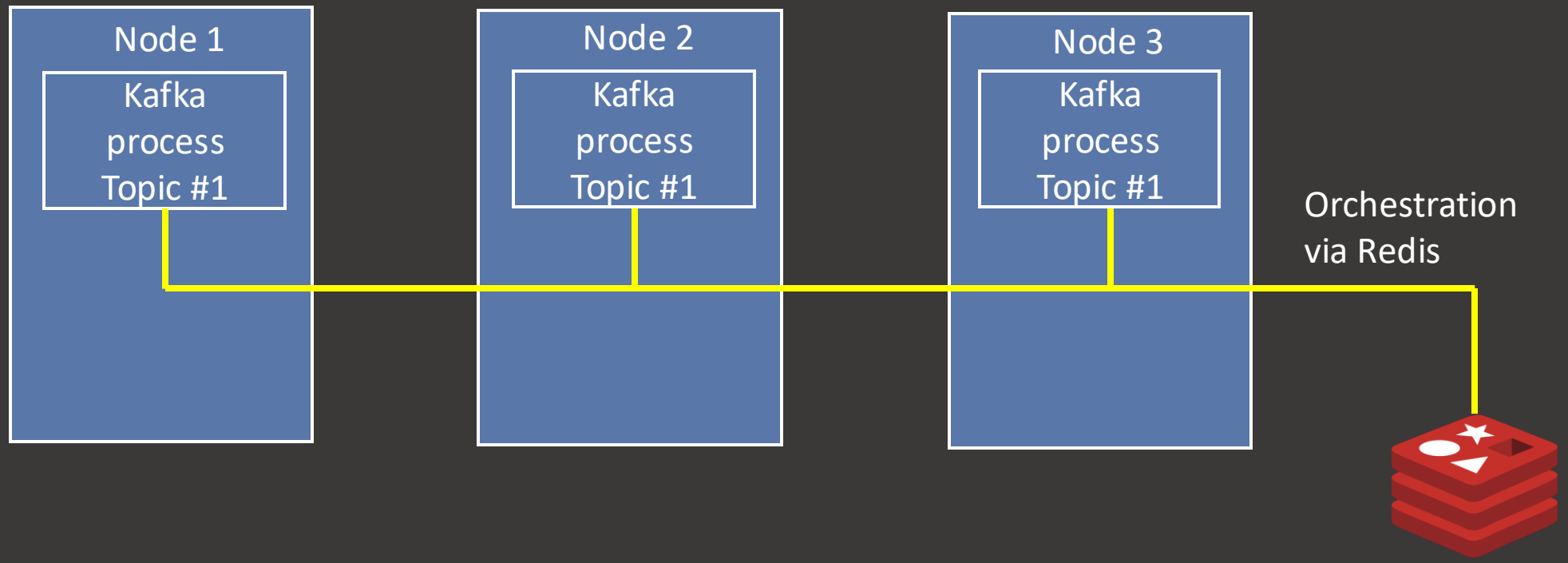
Processes / Managed / Concurrent



Admin

```
Config {
  workers: 4,
  status: stopped
}
```

Processes / Managed / Concurrent



sebastian.dehne@systek.no

Processes / Managed / Concurrent


```
ClusterState {
  nodes: [
    {
      id: node1,
      activeWorkers: 0,
      heartbeat: <timestamp>
    }
  ]
}
```

```
Config {
  workers: 4,
  status: stopped
}
```

sebastian.dehne@systek.no

Skatteetaten

Systek

Processes / Managed / Concurrent


```
ClusterState {
  nodes: [
    {
      id: node1,
      activeWorkers: 0,
      heartbeat: <timestamp>
    }
  ]
}
```

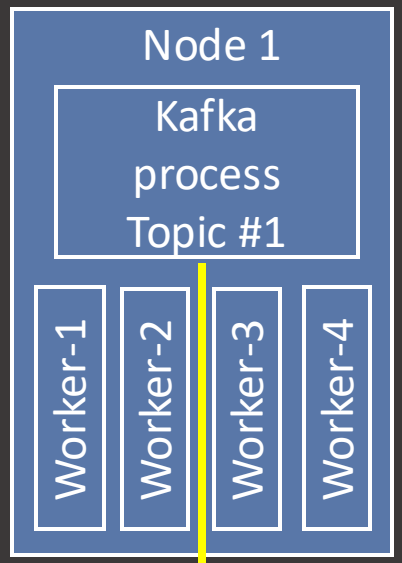
```
Config {
  workers: 4,
  status: running
}
```

sebastian.dehne@systek.no

Skatteetaten

Systek

Processes / Managed / Concurrent




```
ClusterState {
  nodes: [
    {
      id: node1,
      activeWorkers: 4,
      heartbeat: <timestamp>
    }
  ]
}
```

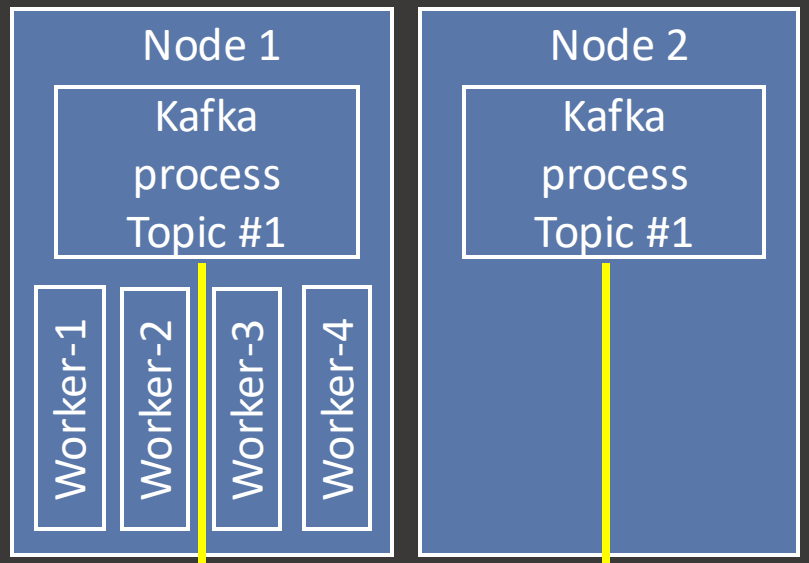
```
Config {
  workers: 4,
  status: running
}
```

sebastian.dehne@systek.no

Skatteetaten

Systek

Processes / Managed / Concurrent




```
ClusterState {
  nodes: [
    {
      id: node1,
      activeWorkers: 4,
      heartbeat: <timestamp>
    },
    {
      id: node2,
      activeWorkers: 0,
      heartbeat: <timestamp>
    }
  ]
}
```

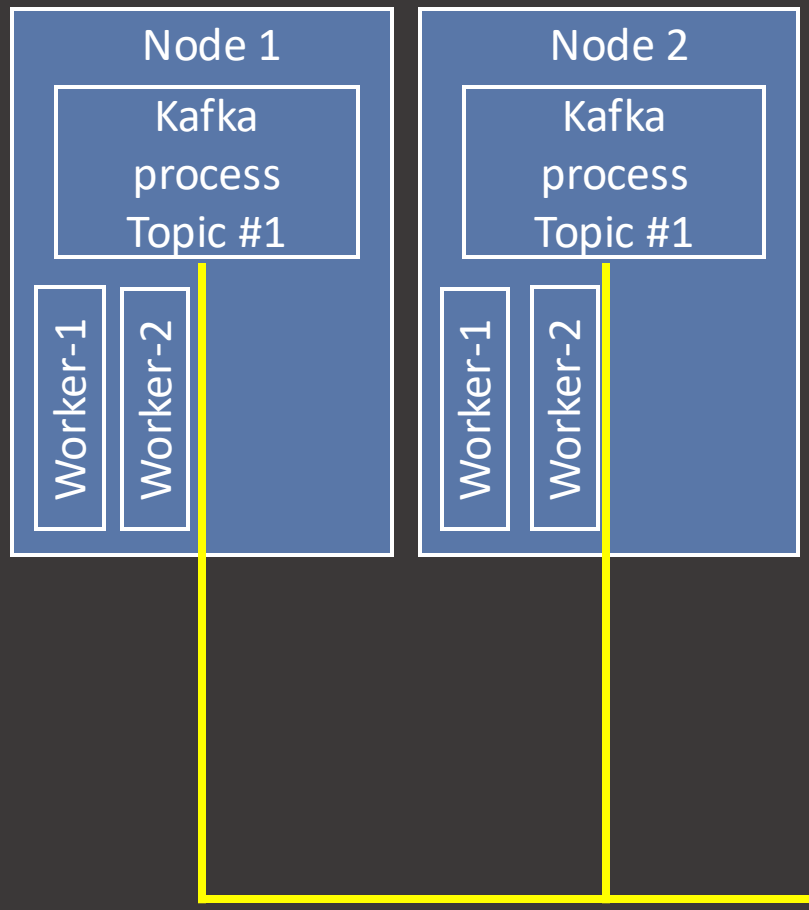
```
Config {
  workers: 4,
  status: running
}
```

sebastian.dehne@systek.no

Skatteetaten

Systek

Processes / Managed / Concurrent




```
ClusterState {
  nodes: [
    {
      id: node1,
      activeWorkers: 2,
      heartbeat: <timestamp>
    },
    {
      id: node2,
      activeWorkers: 2,
      heartbeat: <timestamp>
    }
  ]
}
```

```
Config {
  workers: 4,
  status: running
}
```

sebastian.dehne@systek.no

Skatteetaten

Systek

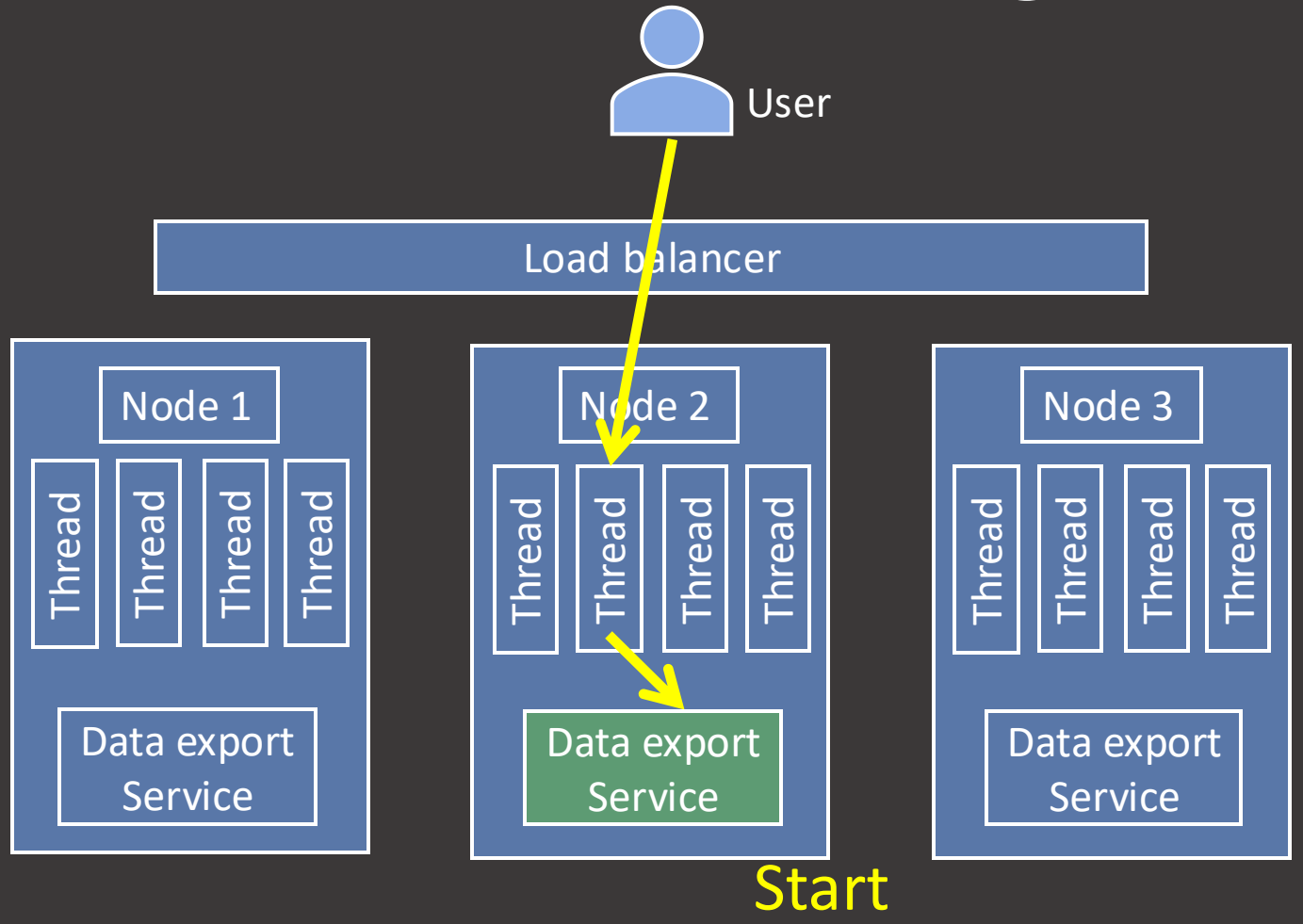
```kotlin
data class ClusterState(
    val nodes: Map<String, NodeState> = emptyMap()
) {
    fun validNeighbors(myNodeName: String) = nodes.values
        .filter { it.isAlive() && it.nodeName != myNodeName }

    fun update(n: NodeState) = copy(
        nodes = nodes.filter { it.value.isAlive() } + (n.nodeName to n))
}

data class NodeState(
    val nodeName: String,
    val heartbeat: Instant,
    val activeWorkers: Int,
)

fun NodeState.isAlive() = heartbeat.plusSeconds( secondsToAdd: 30).isAfter(Instant.now())
```

```
42    private fun readWriteModify(processName: String, fn: (existing: ClusterState) -> ClusterState?) {
43        val key = namespaceKey(processName)
44
45        for (i in 0 ≤ .. ≤ 100) {
46            jedis.watch(key)
47
48            val existing = jedis.get(key)?.let {
49                objectMapper.readValue(it)
50            } ?: ClusterState()
51
52            val updated = fn(existing) ?: return
53
54            val ok = jedis.multi().let { tx ->
55                tx.set(key, objectMapper.writeValueAsString(updated))
56                tx.exec().size == 1
57            }
58
59            if (ok) return
60        }
61        error("Too many retries")
62    }
```

```
14      val myNodeName = System.getenv( name: "NODE_NAME")
15      private val workers: List<Worker> = mutableListOf()
16
17      fun heartBeat() { // runs every 10 seconds
18          var startOrStopWorkers = 0
19          val config = getConfig()
20
21          readWriteModify(processName) { clusterState ->
22              startOrStopWorkers = calculateWorkers(
23                  need = if (config.running) config.workers else 0,
24                  activeRemote = clusterState.validNeighbors(myNodeName).sumOf { it.activeWorkers },
25                  activeLocally = workers.size
26              )
27
28              clusterState.update(
29                  NodeState(
30                      nodeName = myNodeName,
31                      heartbeat = Instant.now(),
32                      activeWorkers = workers.size
33                  )
34              )
35          }
36
37          if (startOrStopWorkers != 0) {
38              startOrStopWorkers(startOrStopWorkers)
39          }
40      }
```

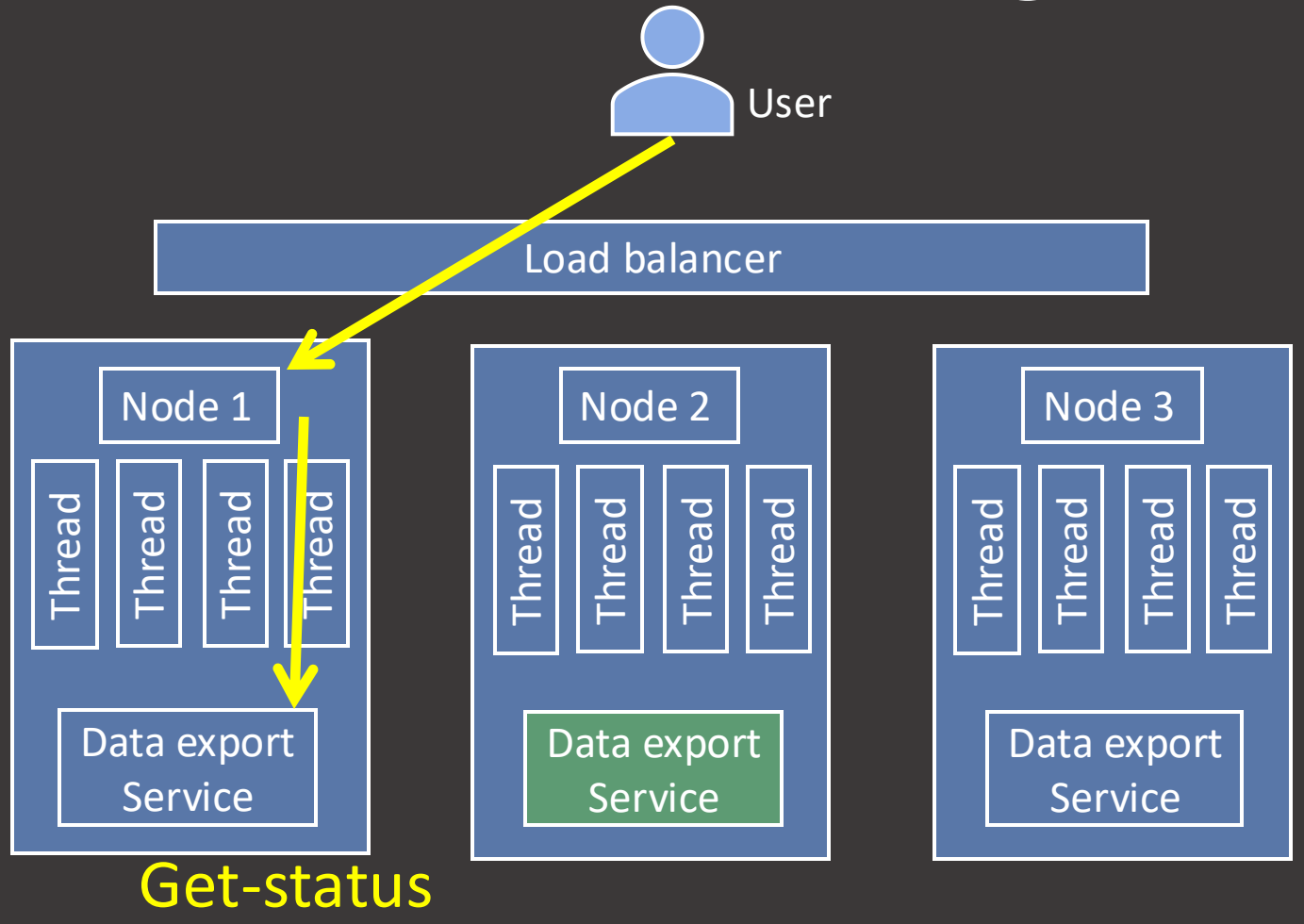
sebastian.dehne@systek.no

Processes / Ad-hoc (unmanaged)

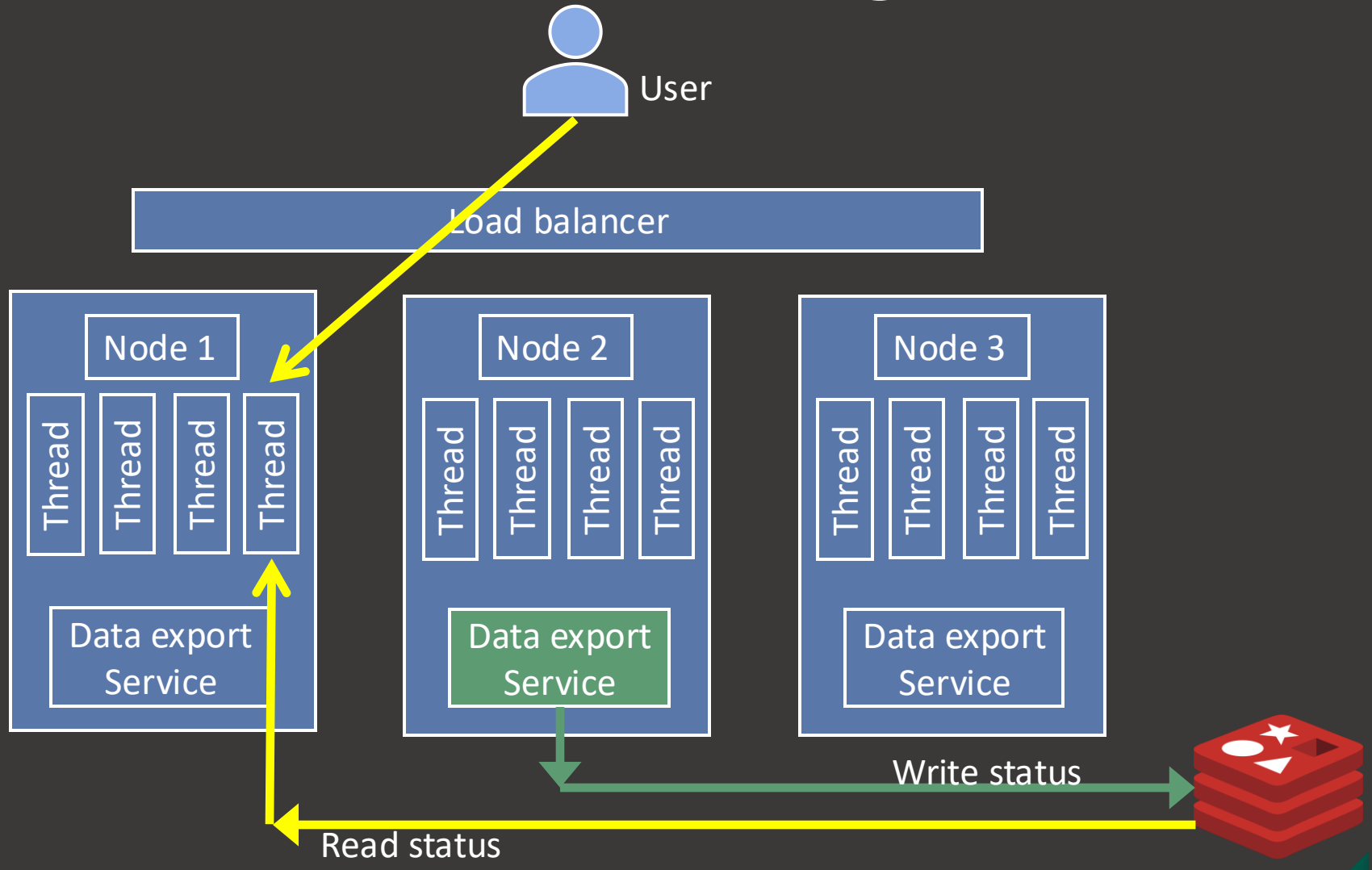


sebastian.dehne@systek.no

Processes / Ad-hoc (unmanaged)



Processes / Ad-hoc (unmanaged)



Skatteetaten

Systek

```kotlin
abstract class AdhocProcess<I, O>(
    private val objectMapper: ObjectMapper,
    private val jedisSource: () -> Jedis,
    private val executorService: ExecutorService,
    private val ttlInSeconds: Long = 300,
) {

    abstract fun inputArgumentsToStableId(inputArguments: I): String
    abstract fun runAdhocTask(inputArguments: I, updateProgress: (progress: String?) -> Unit): O

    // Polled by the client
    fun startOrGetStatus(inputArguments: I, restartIfNotRunning: Boolean = false): AdhocProcessState<O>? {...}

    private fun start(id: String, inputArguments: I) {...}
    private fun readModifyWrite(id: String, fn: (existing: AdhocProcessState<O>?) -> AdhocProcessState<O>?): Boolean {...}
    private fun namespaceKey(id: String) = "${this::class.java.name}-$id"
}

data class AdhocProcessState<T>(
    val id: String,
    val startedAt: Instant = Instant.now(),
    val stoppedAt: Instant? = null,
    val progress: String? = null,
    val error: String? = null,
    val result: T? = null
)
```

```
20      // Polled by the client
21      fun startOrGetStatus(inputArguments: I, restartIfNotRunning: Boolean = false): AdhocProcessState<O>? {
22          var start = false
23          val id = inputArgumentsToStableId(inputArguments)
24
25          var status: AdhocProcessState<O>? = AdhocProcessState(id)
26
27          readModifyWrite(id) { existing ->
28              if ((existing == null || existing.stoppedAt != null) && restartIfNotRunning) {
29                  start = true
30                  status
31              } else {
32                  status = existing
33                  null
34              }
35          }
36
37          if (start) {
38              executorService.submit { start(id, inputArguments) }
39          }
40
41          return status
42      }
```

sebastian.dehne@systek.no

Agenda

- Project SIRIUS
- Locking with Redis
- Running processes with Redis
- **Other Redis use-cases**
- Conclusion / lessions learned

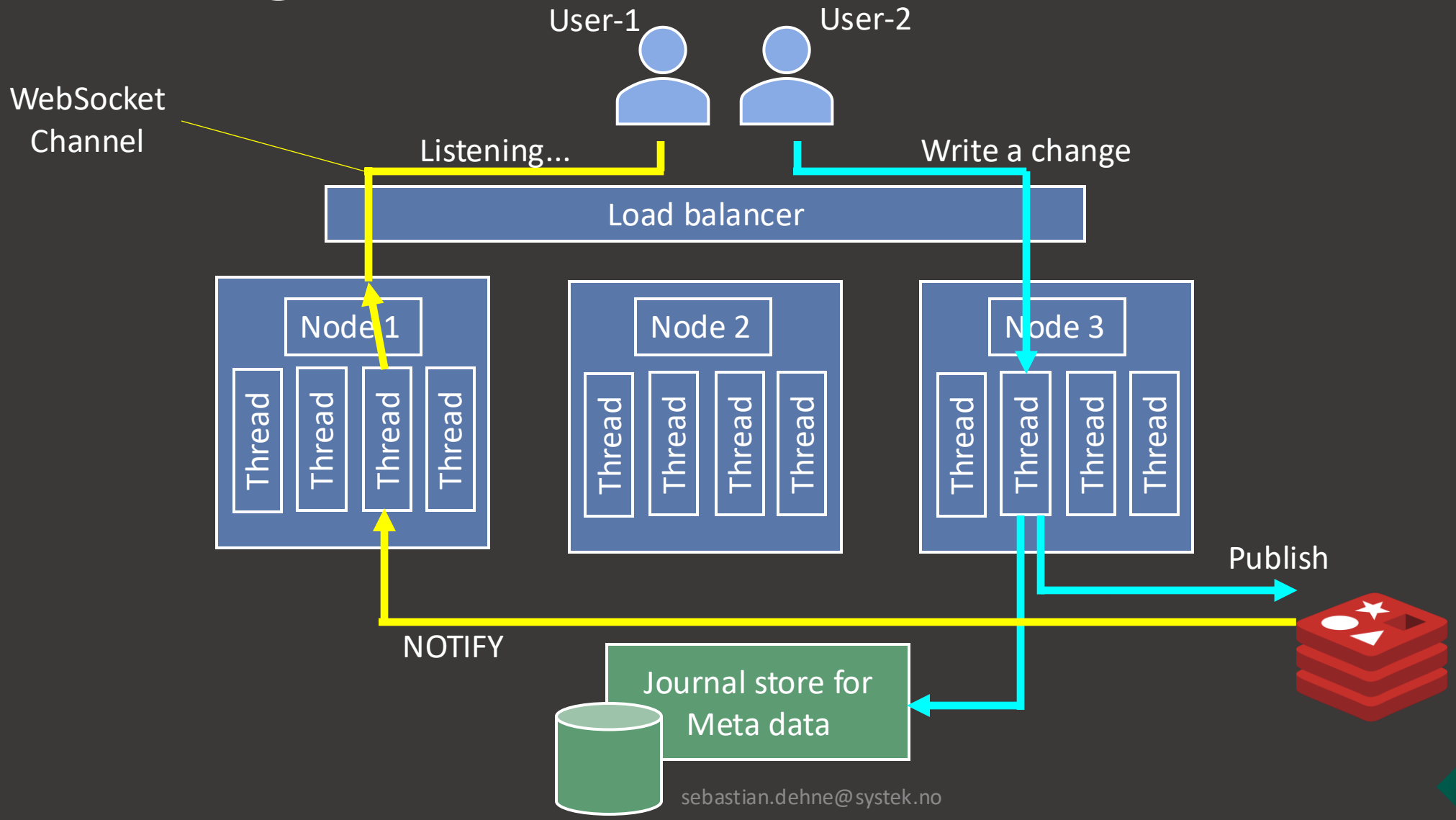
https://github.com/sebdehne/redis-at-skatt-talk

Skatteetaten

sebastian.dehne@systek.no

Systek

Pushing events to frontend



sebastian.dehne@systek.no

Skatteetaten

Systek

Download dumps from a *specific instance*

- Need to fetch a thread-dump or a heap-dump from a specific instance
- Have all instances publish their IP-addr to Redis
- If a fetch-request is not for the local instance
  1. Fetch the IP for the target instance
  2. Proxy the request to that target instance
     (remember to stream the response!)

sebastian.dehne@systek.no

Other use cases we use Redis for…

- Distribute configuration changes at runtime
  - Feature toggles
- Invalidate caches across instances

sebastian.dehne@systek.no

Conclusion / lessions learned

- On paper, building locks on Redis (single instance) is *not* “fault tolerant”
- But in practice, Redis is the most *stable* component in the entire project
- Redis is more stable than Kafka, which runs as a “*HA-cluster*”
- We added metrics, but nobody cares about them
- Handles many thousands reads/writes per second
- Nobody talks about Redis (in our team), because it always just works
- It has been a real enabler to solve many problems related to multi-node deployments

sebastian.dehne@systek.no

End

Thank you for your attention

https://github.com/sebdehne/redis-at-skatt-talk

Skatteetaten

sebastian.dehne@systek.no

Systek