identifying design flaws in a lock-free task pool with tla+

Vasil Dyadov · Alexander Kogtenkov · Ilya Shchepetkov

Kaspersky Lab

ABZ 2026 · 東京 · May 18 — May 20

about me

- I am Lead Research Developer in the Formal Methods Group at Kaspersky
- We work on KasperskyOS: a general-purpose microkernel-based OS
- We focus mainly on applying state-based formal methods during the design phase of development

microkernel and ipc

Inter-process communication is an integral part of the microkernel

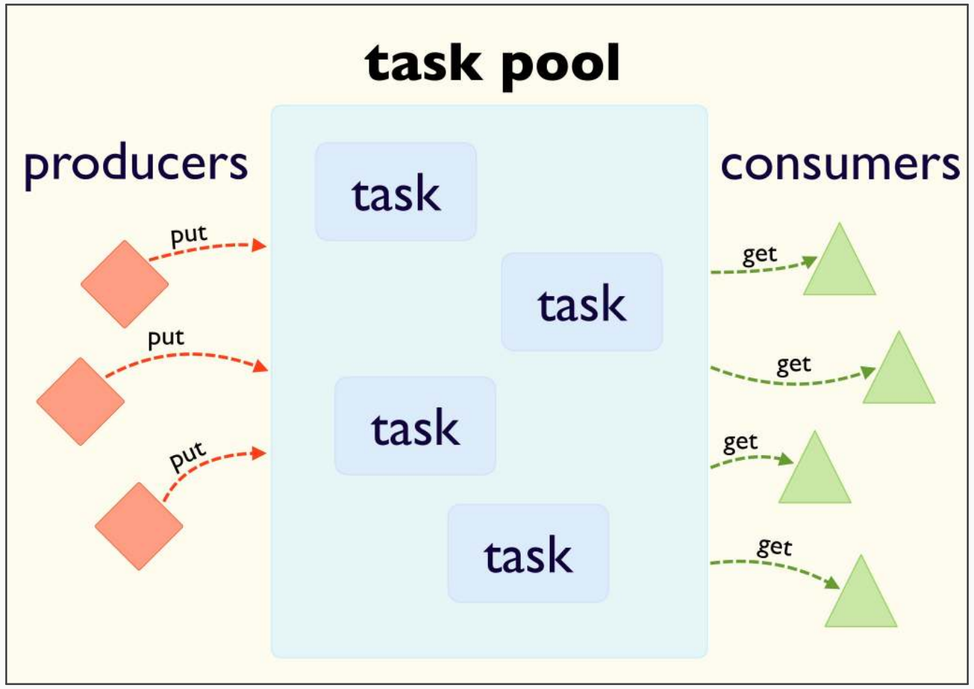
It should be fast, secure and correct

meet salsa

Scalable And Low Synchronization Algorithm
for Producer-Consumer Task Pools

meet salsa

Scalable And Low Synchronization Algorithm
for Producer-Consumer Task Pools



salsa

- Scales linearly with the number of threads, outperforms other alternatives
- Lock-free algorithm without strong atomic operations or memory barriers in the fast path

What is the catch here?

the catch

Concurrent data structures are notoriously difficult
to design and implement

the solution

Formal Methods

our plan

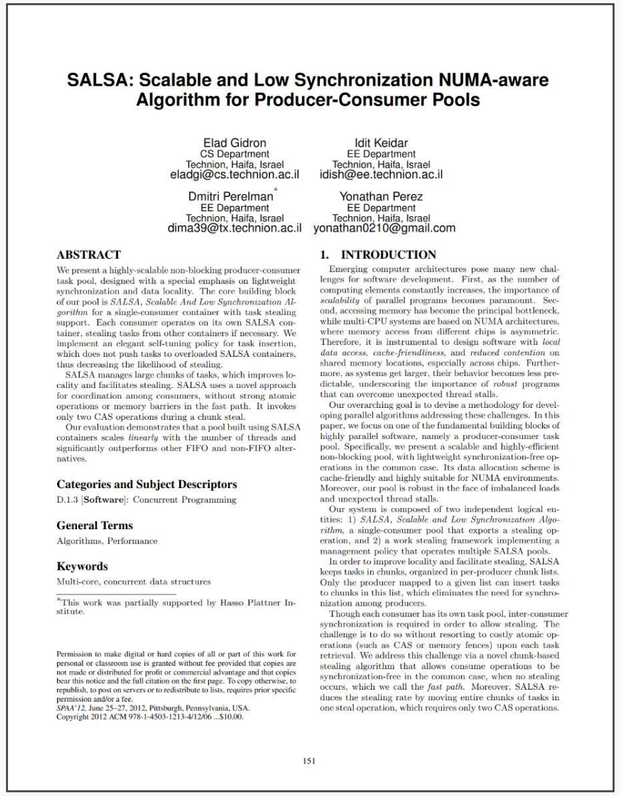
1. Understand the algorithm
2. Formalize and verify it
3. Deliver the resulted artifacts to the developers
4. Verify the conformance between specification and implementation

long story short

1. Understand the algorithm
2. Formalize and verify it ← found design errors in the spec
3. Deliver the resulted artifacts to the developers
4. Verify the conformance between specification and implementation

understanding

- SALSA is presented in in the peer-reviewed paper*
- There is a separate technical report
- There is no publicly available implementation



SALSA: Scalable and Low Synchronization NUMA-aware Algorithm for Producer-Consumer Pools

Elad Gidron
CS Department
Technion, Haifa, Israel
eladgi@cs.technion.ac.il

Idit Keidar
EE Department
Technion, Haifa, Israel
idish@ee.technion.ac.il

Dmitri Perelman*
EE Department
Technion, Haifa, Israel
dima39@tx.technion.ac.il

Yonathan Perez
EE Department
Technion, Haifa, Israel
yonathan0210@gmail.com

ABSTRACT

We present a highly-scalable non-blocking producer-consumer task pool, designed with a special emphasis on lightweight synchronization and data locality. The core building block of our pool is *SALSA, Scalable And Low Synchronization Algorithm* for a single-consumer container with task stealing support. Each consumer operates on its own SALSA container, stealing tasks from other containers if necessary. We implement an elegant self-tuning policy for task insertion, which does not push tasks to overloaded SALSA containers, thus decreasing the likelihood of stealing.

SALSA manages large chunks of tasks, which improves locality and facilitates stealing. SALSA uses a novel approach for coordination among consumers, without strong atomic operations or memory barriers in the fast path. It invokes only two CAS operations during a chunk steal.

Our evaluation demonstrates that a pool built using SALSA containers scales *linearly* with the number of threads and significantly outperforms other FIFO and non-FIFO alternatives.

Categories and Subject Descriptors

D.1.3 [**Software**]: Concurrent Programming

General Terms

Algorithms, Performance

Keywords

Multi-core, concurrent data structures

*This work was partially supported by Hasso Plattner Institute.

SPAA'12, June 25–27, 2012, Pittsburgh, Pennsylvania, USA.

1. INTRODUCTION

Emerging computer architectures pose many new challenges for software development. First, as the number of computing elements constantly increases, the importance of *scalability* of parallel programs becomes paramount. Second, accessing memory has become the principal bottleneck, while multi-CPU systems are based on NUMA architectures, where memory access from different chips is asymmetric. Therefore, it is instrumental to design software with *local data access*, *cache-friendliness*, and *reduced contention* on shared memory locations, especially across chips. Furthermore, as systems get larger, their behavior becomes less predictable, underscoring the importance of *robust* programs that can overcome unexpected thread stalls.

Our overarching goal is to devise a methodology for developing parallel algorithms addressing these challenges. In this paper, we focus on one of the fundamental building blocks of highly parallel software, namely a producer-consumer task pool. Specifically, we present a scalable and highly-efficient non-blocking pool, with lightweight synchronization-free operations in the common case. Its data allocation scheme is cache-friendly and highly suitable for NUMA environments. Moreover, our pool is robust in the face of imbalanced loads and unexpected thread stalls.

Our system is composed of two independent logical entities: 1) *SALSA, Scalable and Low Synchronization Algorithm*, a single-consumer pool that exports a stealing operation, and 2) a work stealing framework implementing a management policy that operates multiple SALSA pools.

In order to improve locality and facilitate stealing, SALSA keeps tasks in chunks, organized in per-producer chunk lists. Only the producer mapped to a given list can insert tasks to chunks in this list, which eliminates the need for synchronization among producers.

Though each consumer has its own task pool, inter-consumer synchronization is required in order to allow stealing. The challenge is to do so without resorting to costly atomic operations (such as CAS or memory fences) upon each task retrieval. We address this challenge via a novel chunk-based stealing algorithm that allows consume operations to be synchronization-free in the common case, when no stealing occurs, which we call the *fast path*. Moreover, SALSA reduces the stealing rate by moving entire chunks of tasks in one steal operation, which requires only two CAS operations.

151

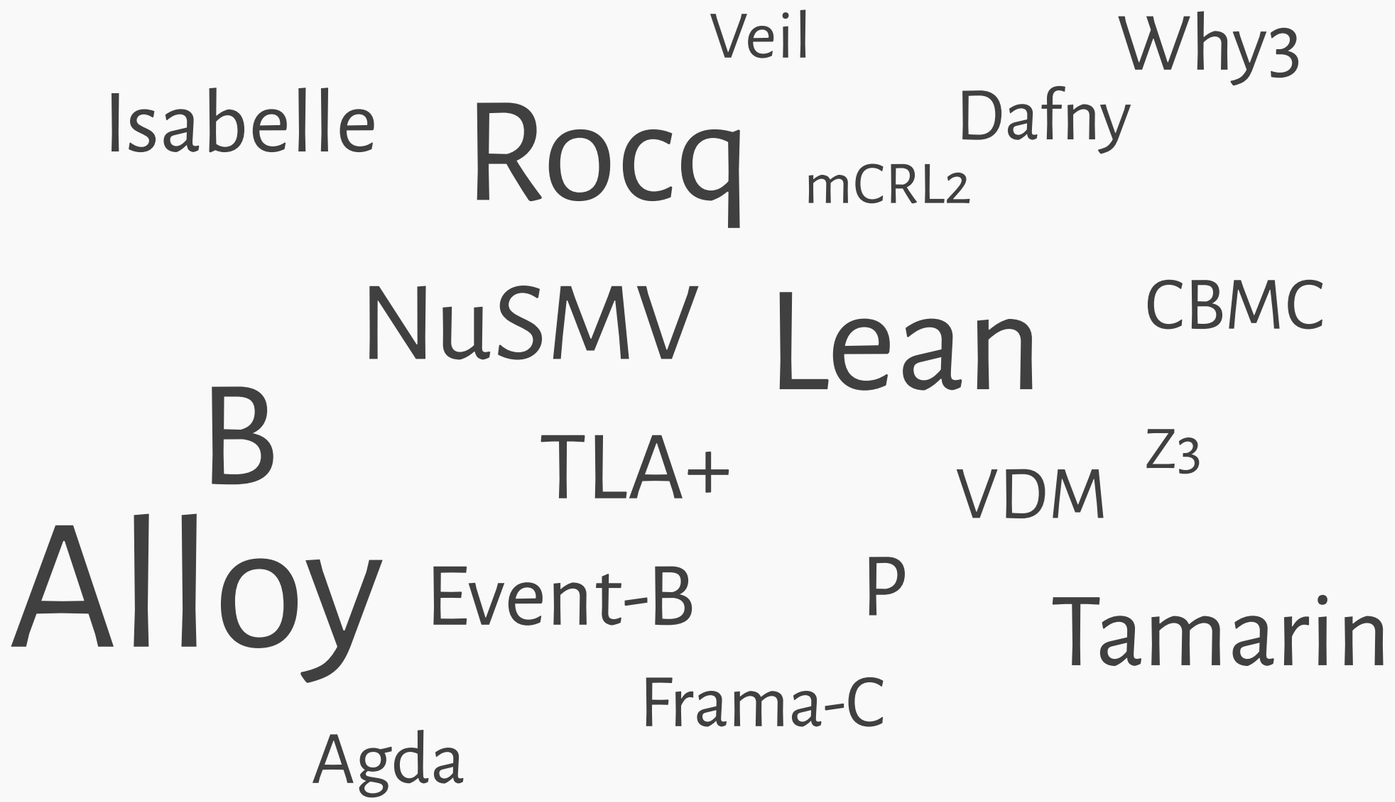
* ACM Symposium on Parallelism in Algorithms and Architectures

architecture

Everything you need to know:

- Producers and Consumers
- Tasks
- Stealing and Chunks
- Chunk pool

which method to choose?



constraints

experience · deadline · results

tla+ + pluscal

TLA+ is a specification language that can be used for modelling concurrent systems as state machines

TLA+ specs can be verified using TLC model checker

PlusCal transpiles to TLA+ and is useful for specifying algorithms

properties

Concurrency:

- Linearizability
- Lock-freedom
- Absence of ABA problems

Common:

- Functional correctness
- Structural correctness

SALSA paper does not provide the list of functional or structural properties

translation

Pseudocode

```
1: Function checkLast(Node n):
2:   if(n.idx + 1 = CHUNK_SIZE) then
3:     n.c ← ⊥;
4:     return chunk to chunkPool;
5:     currentNode ← ⊥;
```

Simplified PlusCal

```
 1: procedure checkLast(n)
 2: variable chunk = NULL;
 3: if nodes[n].idx + 1 = CHUNK_SIZE then
 4:   chunk := nodes[n].c;
 5:   nodes[n].c := NULL;
 6:   if chunk /= NULL then
 7:     scPools[self].chunkPool := Append(
 8:       scPools[self].chunkPool, chunk);
 9:   end if;
10:   scPools[self].currentNodeId := NULL;
11: end if;
```

We need to interpret over-abstract parts of the pseudocode and to define atomicity zones

specification

```
2063  \* Chunk pools of different consumers should not contain the same chunk at the
2064  \* same time.
2065  \* ERROR: this invariant does not hold.
2066  PoolsDoNotIntersect ==
2067      \A c \in CONSUMERS, cc \in CONSUMERS: c /= cc =>
2068          Range(scPools[c].chunkPool) \intersect Range(scPools[cc].chunkPool) = {}
2069
2070  \* Producers should not write tasks into a chunk that was returned to the
2071  \* chunk pool.
2072  ProducerCantWriteIntoPool ==
2073      \A p \in PRODUCERS: pc[p] = "insertFinish" =>
2074          \A c \in CONSUMERS: prodChunkId[p] \notin Range(scPools[c].chunkPool)
2075
2076  \* Two producers should not write tasks to the same chunk.
2077  ProducerCantWriteToTheSameChunk ==
2078      \A p1 \in PRODUCERS, p2 \in PRODUCERS:
2079          (p1 /= p2 /\ prodChunkId[p1] /= NULL) =>
2080              prodChunkId[p1] /= prodChunkId[p2]
```

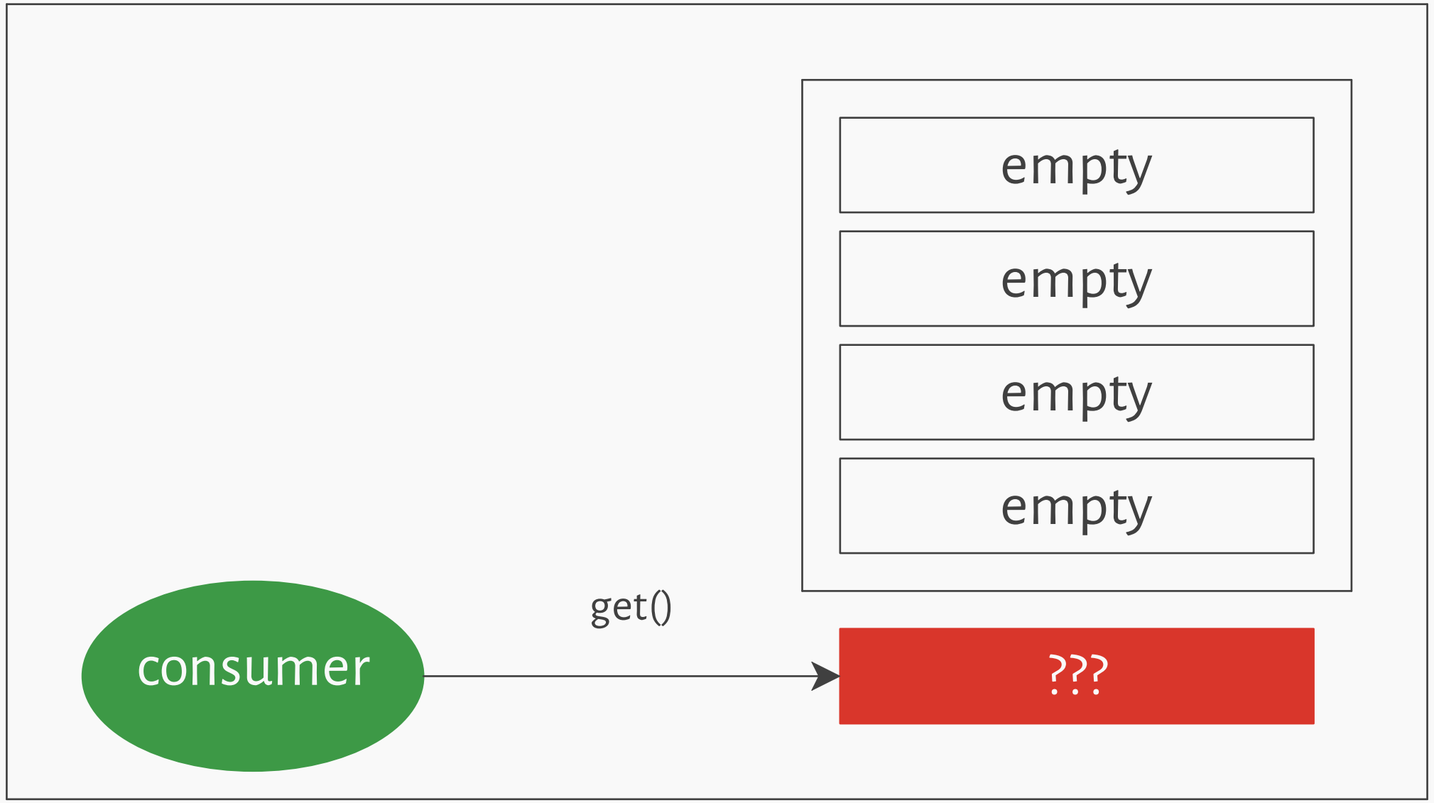
verification

This is where we started finding issues

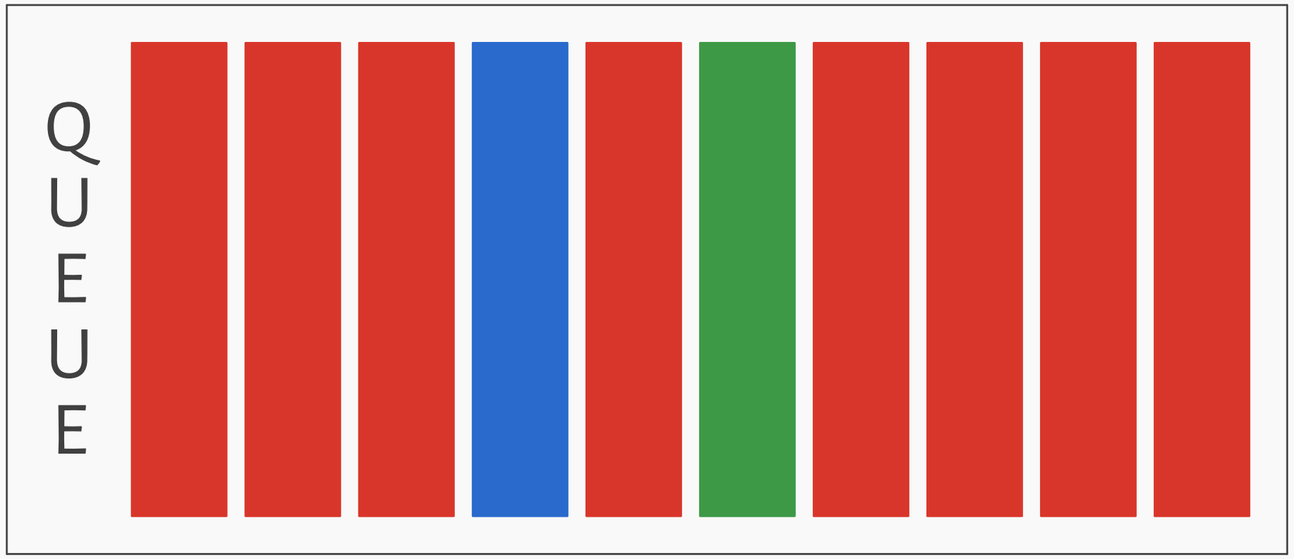
First batch of error traces was the usual one: false positives

The next ones looked real

index out of bounds error

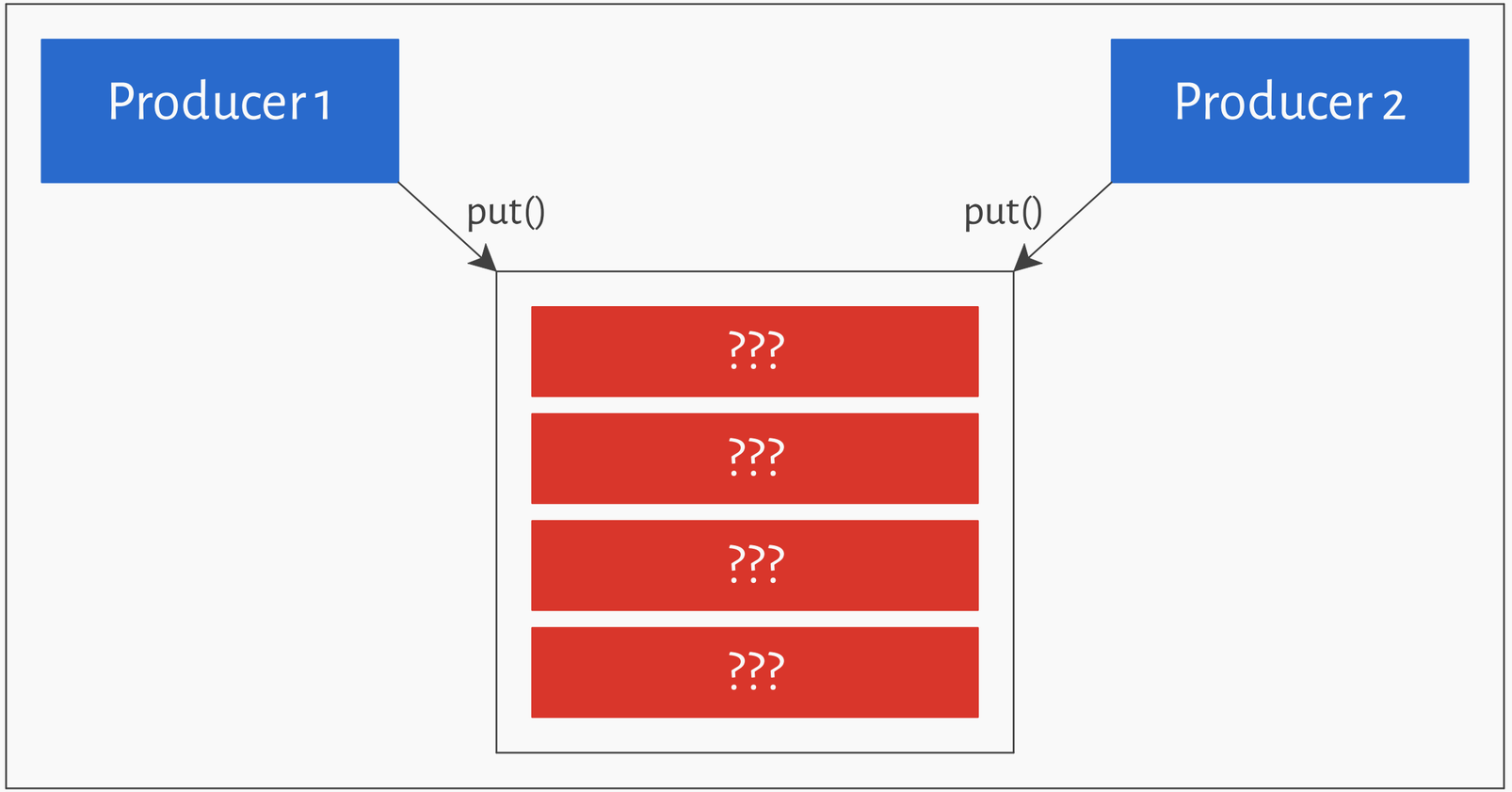


resource mismanagement



Where  rectangles are pointers to the same object

total breakdown



task loss

are these issues real?

1. Specification and errors traces were independently audited by a separate person from our team
2. We tried different configurations
3. We searched for other SALSA reports

Caveat: we are not domain experts

stats

- 850 lines of invariants and PlusCal code
- All found error traces are around 100 steps of execution
- A couple of seconds until an invariant violation
  - using 16 core Ryzen 5950x CPU
- 2 person-months of work

contribution

We formally specified SALSA and found a number of issues in its design

Specification is publicly available and can be used to fix the issues: https://codeberg.org/17451k/salsa-spec

Thank you!