Composing State Machine Replicas

Odorico Machado Mendizabal

Workshop Suíça–Brasil: Um Olhar Atual sobre Sistemas Distribuídos: Da Pesquisa à Aplicação no Mundo Real

15 e 16 de abril de 2025

Breaking the Ice

- **2003** – Bachelor's Degree in Computer Engineering (FURG)
- **2006** – Master's Degree in Computer Science (PUCRS)
- **2007** – SW Developer (HP) SW Performance Analyst (Dell)
- **2011** – Assistant Professor at FURG
- **2016** – PhD in Computer Science
- **2018** – Assistant Professor at UFSC
- **2025** – To be continued…

Research Interest and Recent Results

Consensus Protocols

Paxos variations
High throughput Paxos: ISS Paxos

State Machine Replication

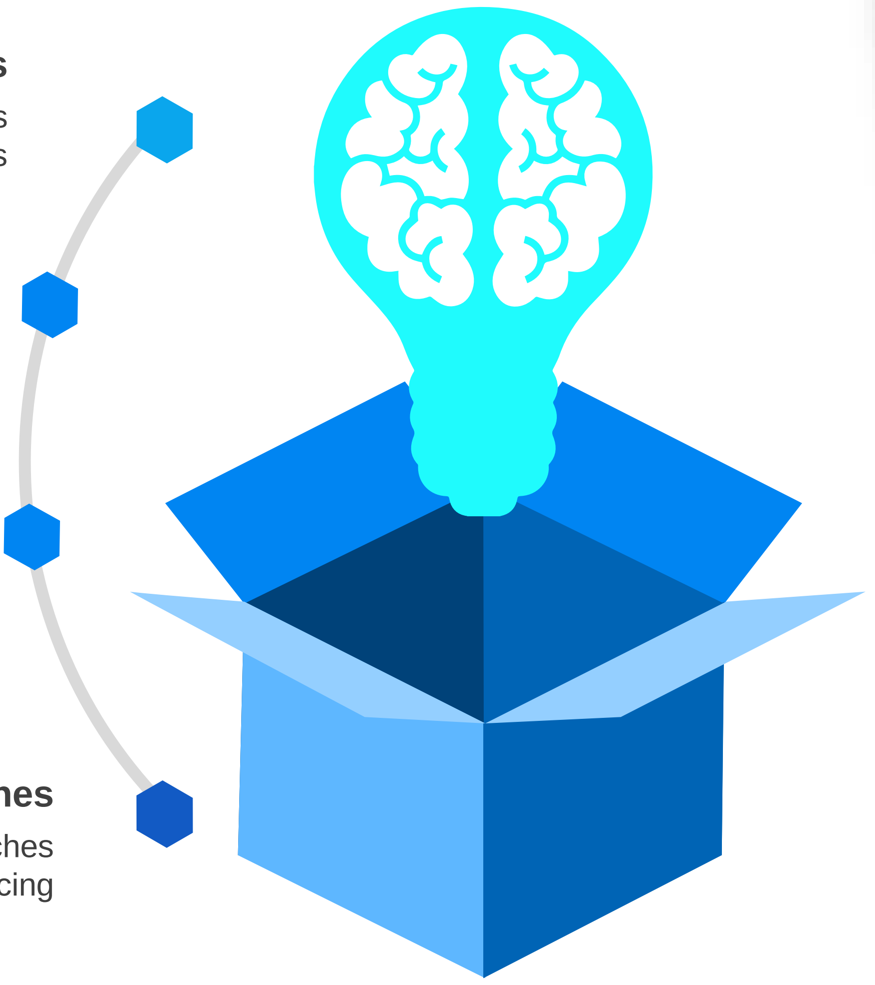
Parallel SMR
SMR composition

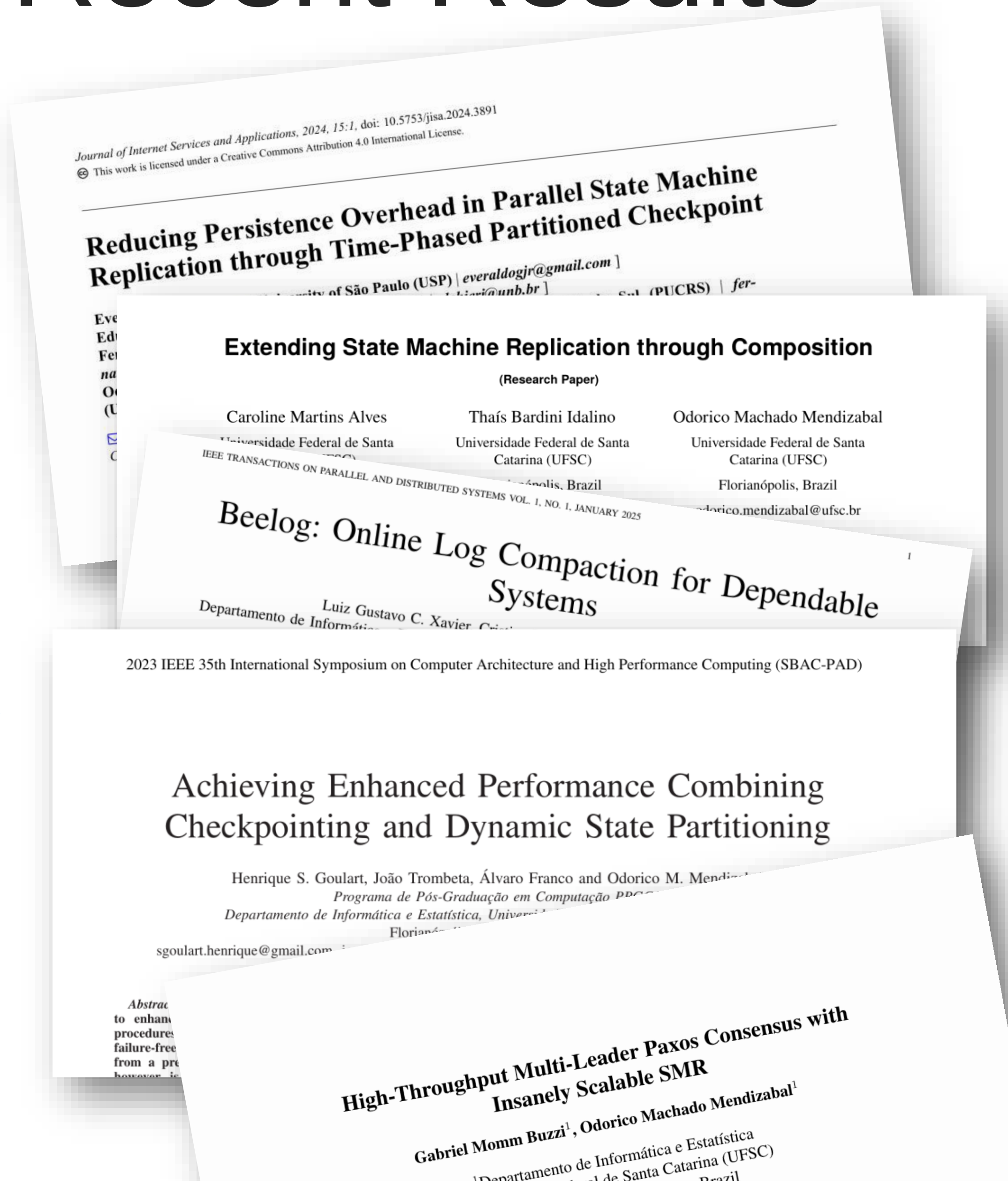
Durability Approaches

Speed up recovery
Log Compaction
Low Overhead Checkpointing/Restore
C/R as a service

Other approaches

Repartitioning approaches
Data Balancing





Journal of Internet Services and Applications, 2024, 15:1, doi: 10.5753/jisa.2024.3891
This work is licensed under a Creative Commons Attribution 4.0 International License.

Reducing Persistence Overhead in Parallel State Machine Replication through Time-Phased Partitioned Checkpoint

ity of São Paulo (USP) | everaldogjr@gmail.com]
biari@unb.br]
Sul (PUCRS) | fer-

Extending State Machine Replication through Composition

(Research Paper)

Caroline Martins Alves — Universidade Federal de Santa
Thaís Bardini Idalino — Universidade Federal de Santa Catarina (UFSC) — polis, Brazil
Odorico Machado Mendizabal — Universidade Federal de Santa Catarina (UFSC) — Florianópolis, Brazil — odorico.mendizabal@ufsc.br

IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS VOL. 1, NO. 1, JANUARY 2025 1

Beelog: Online Log Compaction for Dependable Systems

Luiz Gustavo C. Xavier, Cri
Departamento de Informáti

2023 IEEE 35th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)

Achieving Enhanced Performance Combining Checkpointing and Dynamic State Partitioning

Henrique S. Goulart, João Trombeta, Álvaro Franco and Odorico M. Mendi
Programa de Pós-Graduação em Computação PPG
Departamento de Informática e Estatística, Univer
Florian
sgoulart.henrique@gmail.com

Abstrac
to enhan
procedure
failure-free
from a pre

High-Throughput Multi-Leader Paxos Consensus with Insanely Scalable SMR

Gabriel Momm Buzzi[1], Odorico Machado Mendizabal[1]
[1]Departamento de Informática e Estatística
al de Santa Catarina (UFSC)
Brazil

State Machine Replication

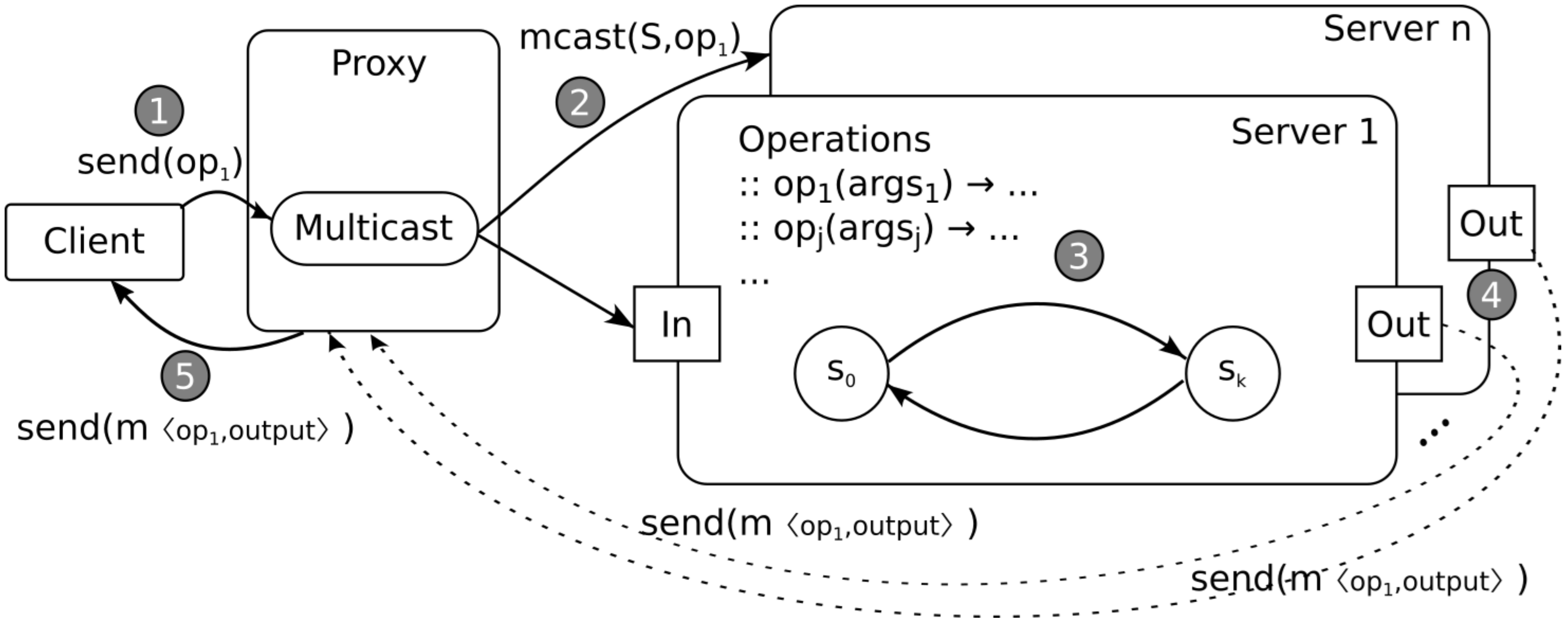
A set of servers behaves as replicated state machines

- Replicas start in the same initial state
- Service operations are deterministic

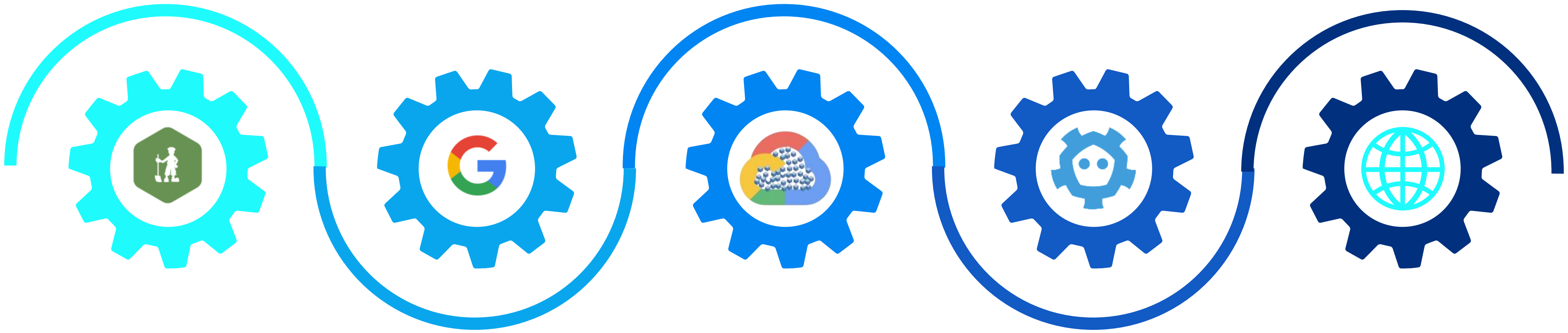
Clients issue commands to every replica through a consensus or atomic broadcast protocol:

- correct replicas receive every command
- if a replica processes a command c_1 before c_2, then no replica process c_2 before c_1

State Machine Replication



State Machine Replication – Applications



Apache Zookeeper

Open source server for highly reliable distributed coordination

Google Chubby

Lock service, used to help coordination in distributed environment using locking semantics

Google File System

Distributed file system that provides efficient and reliable data access

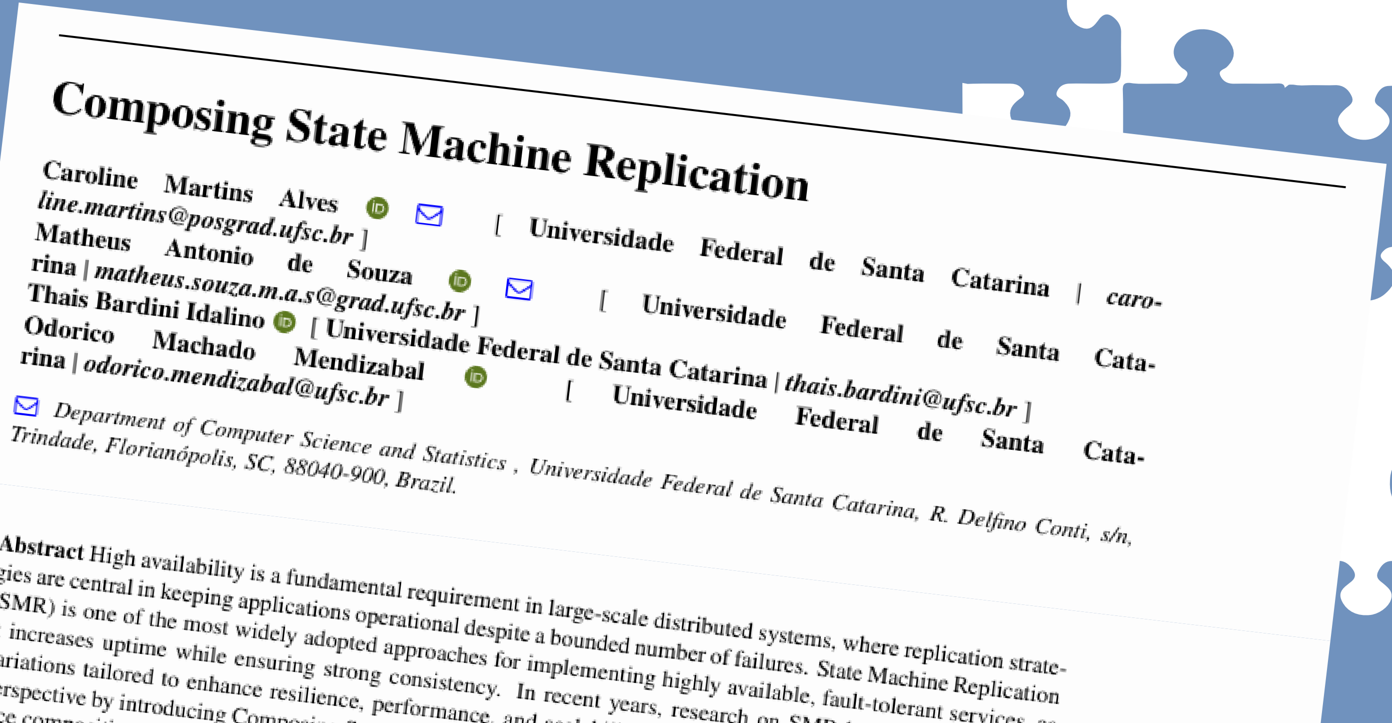
Key-value store services

Ex. etcd, used to replicate metadata on Kubernetes cluster managers

Online services

This talk

“Over the past decades, SMR has gained popularity, leading to extensive research that has enhanced its resilience, performance, and scalability. However, one aspect not yet addressed in SMR is service composition



Composing State Machine Replication

Caroline Martins Alves [Universidade Federal de Santa Catarina | *caroline.martins@posgrad.ufsc.br*]
Matheus Antonio de Souza [Universidade Federal de Santa Catarina | *matheus.souza.m.a.s@grad.ufsc.br*]
Thais Bardini Idalino [Universidade Federal de Santa Catarina | *thais.bardini@ufsc.br*]
Odorico Machado Mendizabal [Universidade Federal de Santa Catarina | *odorico.mendizabal@ufsc.br*]

Department of Computer Science and Statistics , Universidade Federal de Santa Catarina, R. Delfino Conti, s/n, Trindade, Florianópolis, SC, 88040-900, Brazil.

Abstract High availability is a fundamental requirement in large-scale distributed systems, where replication strate-
gies are central in keeping applications operational despite a bounded number of failures. State Machine Replication
SMR) is one of the most widely adopted approaches for implementing highly available, fault-tolerant services,
increases uptime while ensuring strong consistency. In recent years, research on SMR
ariations tailored to enhance resilience, performance, and
erspective by introducing Compo

SMR Research

A (very) quick view of approaches over the years

SMR – other approaches

Byzantine faults

Adding protocols capable of tolerating arbitrary faults

Recovery and Reconfiguration

Aimed to improve resilience

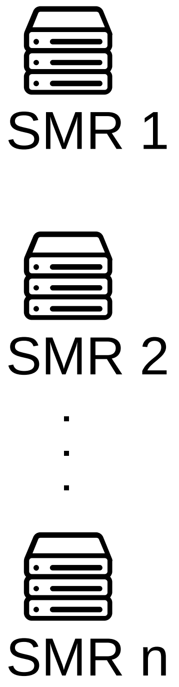
Parallel SMR

Independent requests executed in parallel

Composing State Machine Replication (CSMR)

Build services by combining separate instances of SMR

- Microservice-based systems
- Flexible and loosely coupled solutions
- Development in a modular way



CSMR – Selected use cases

SMR 1 – Lock service

```
boolean acquire (string key)
boolean release (string key)
```

SMR 2 – key-value store

```
string get(string key)
void put(string key, string value)
```

SMR 3 – Logging service

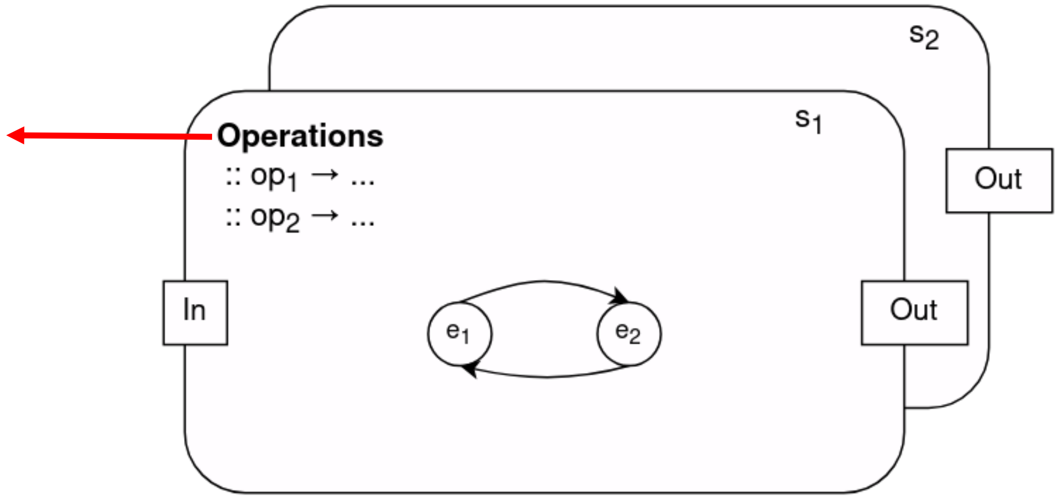
```
void append(Object entry)
Object[] retrieve(int first, int last)
boolean truncate(int index)
```

SMR Formalization: Definitions

- Replicated service
- Operations (with arguments)
- Execution
- Output

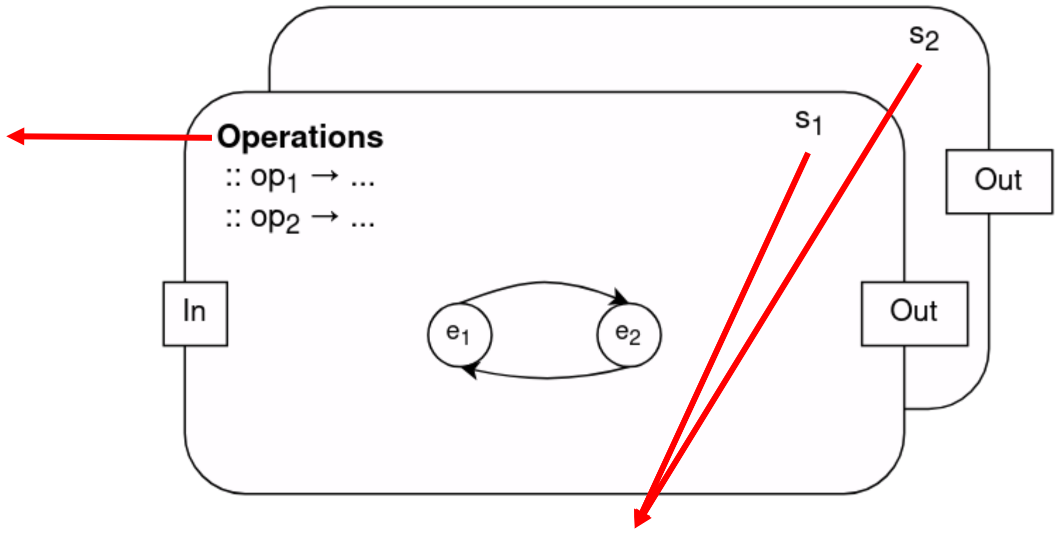
SMR Formalization: Definitions

$O = \{op_1, op_2\}$



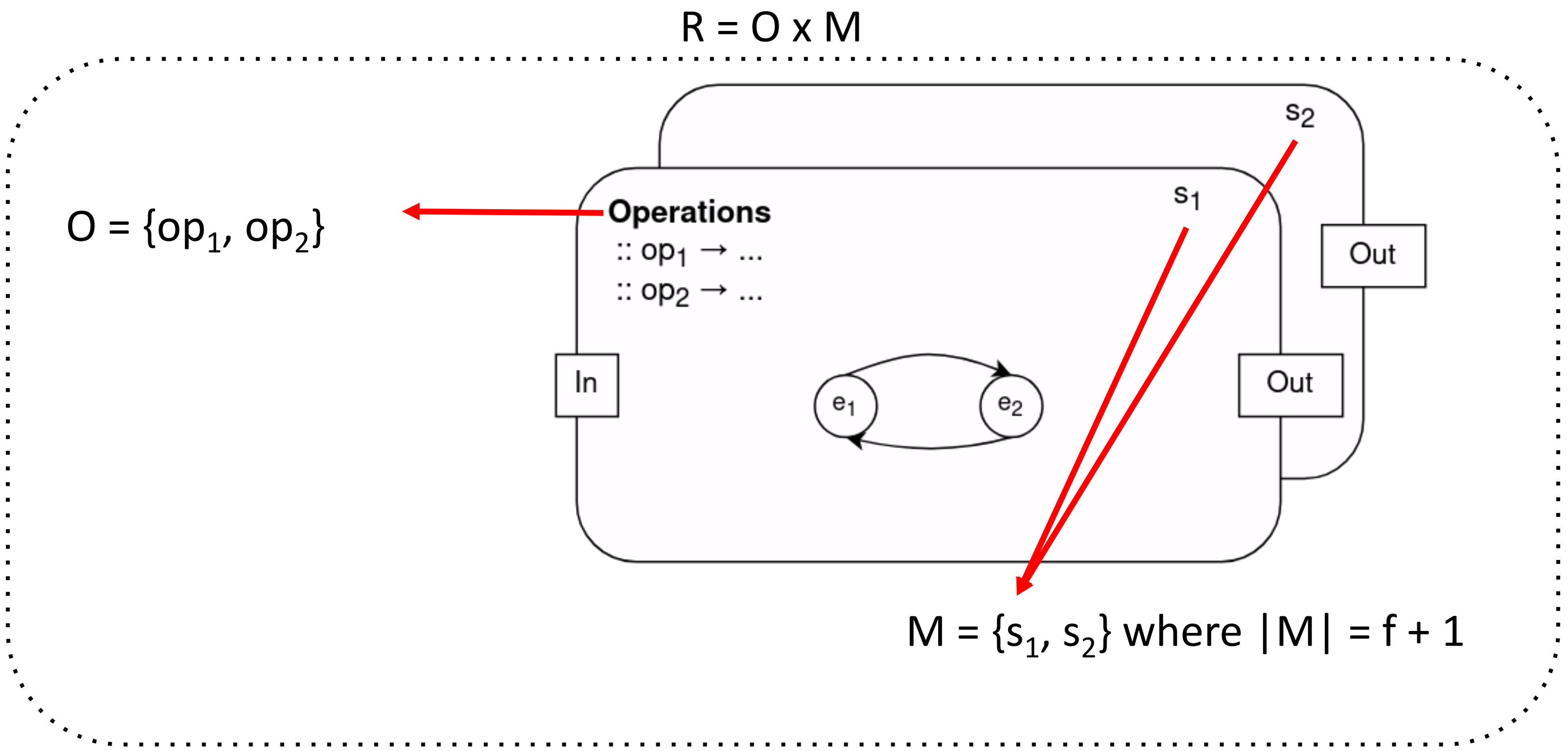
SMR Formalization: Definitions

$O = \{op_1, op_2\}$



$M = \{s_1, s_2\}$ where $|M| = f + 1$

SMR Formalization: Definitions



SMR Formalization: Definitions

- **Definition 1** - Replicated service
  - **Example: SMR 2 - Key-value store**

SMR 2 - Key-value store

O = {get, put}

SMR Formalization: Definitions

- **Definition 1** - Replicated service
  - **Example: SMR 2 - Key-value store**

SMR 2 - Key-value store

$O = \{get, put\}$

Replicas

s_1 s_2

$M = \{s_1, s_2\}$

SMR Formalization: Definitions

- **Definition 1** - Replicated service
  - **Example: SMR 2 - Key-value store**

SMR 2 - Key-value store

$O = \{get, put\}$

Replicas

s_1 s_2

$M = \{s_1, s_2\}$

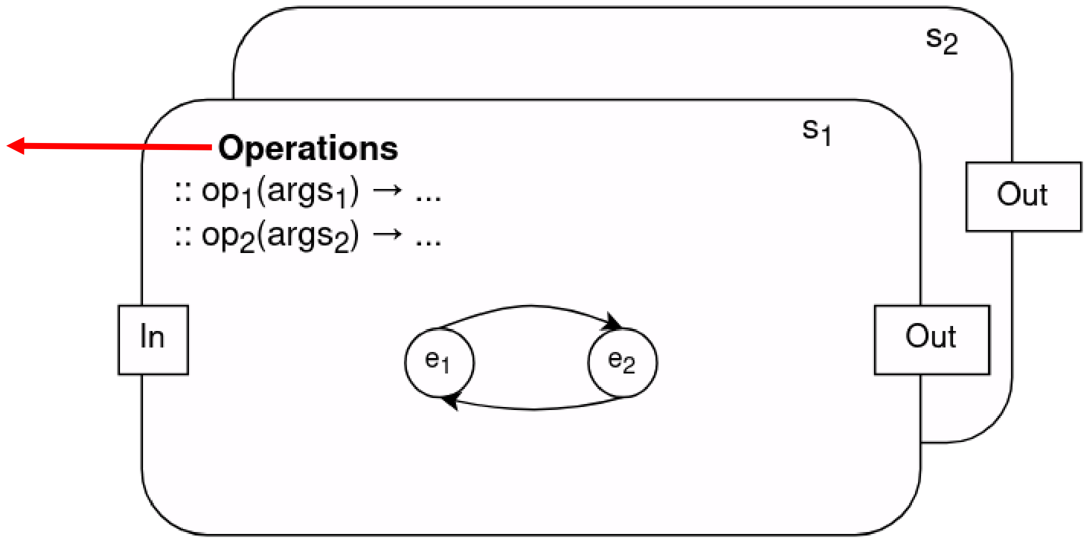
Replicated Service

$R = \{(get, s_1), (get, s_2), (put, s_1), (put, s_2)\}$

SMR Formalization: Definitions

- **Definition 2** - Operations with arguments

$O = \{op_1, op_2\}$

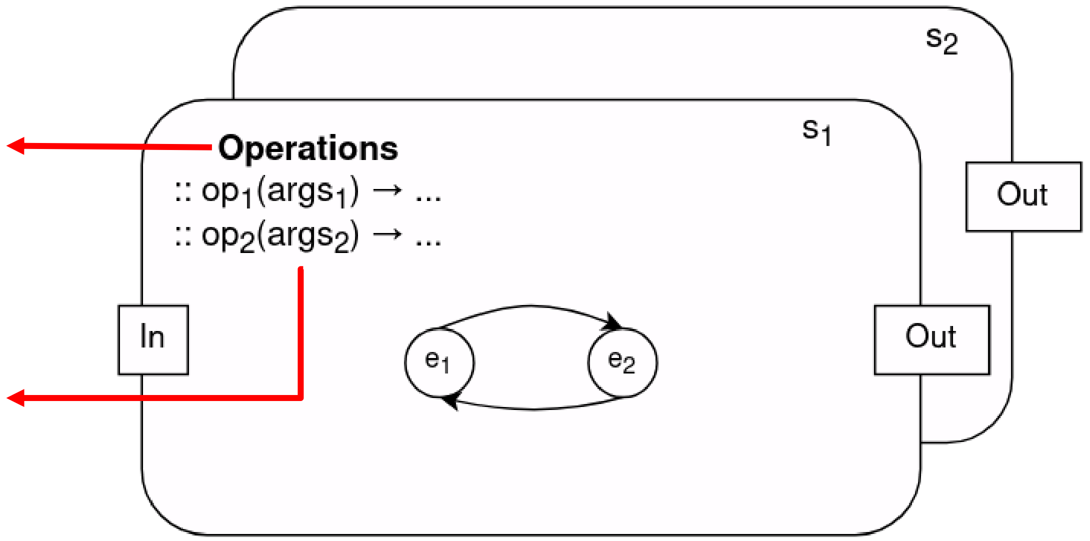


SMR Formalization: Definitions

- **Definition 2** - Operations with arguments

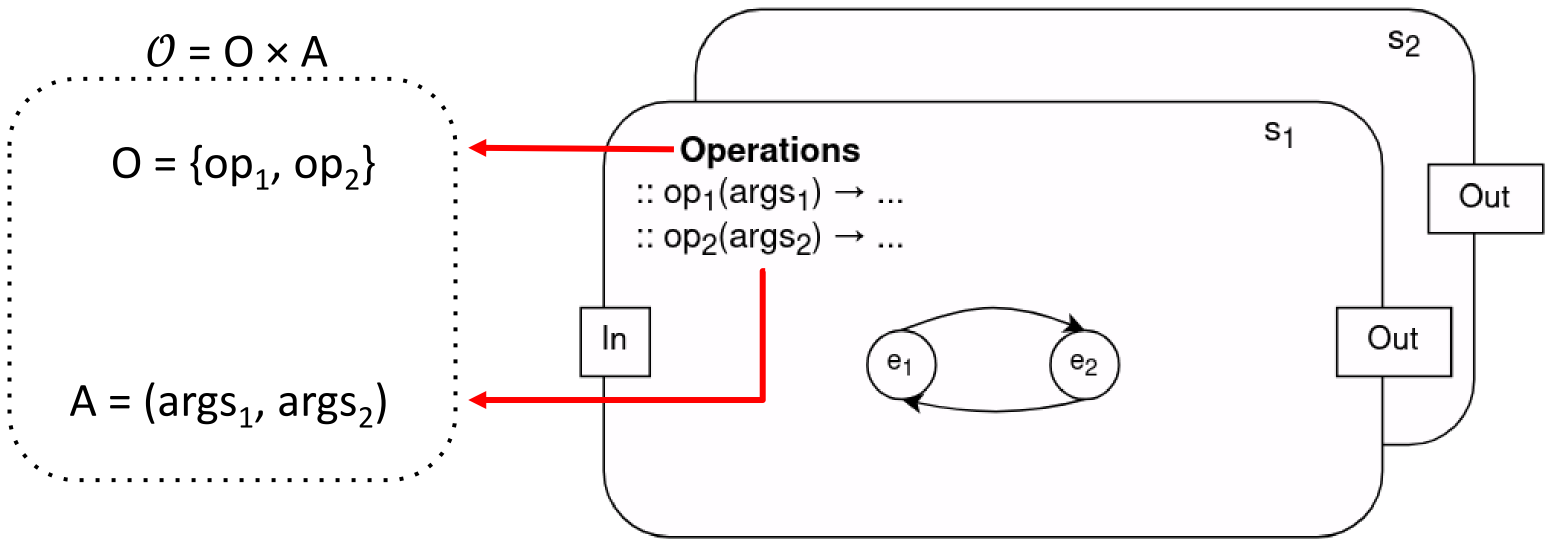
$O = \{op_1, op_2\}$

$A = (args_1, args_2)$



SMR Formalization: Definitions

- **Definition 2** - Operations with arguments



SMR Formalization: Definitions

- **Definition 2** - Operations with arguments
  - **Example: SMR 2 - Key-value store**

SMR 2 - Key-value store

O = {get, put}

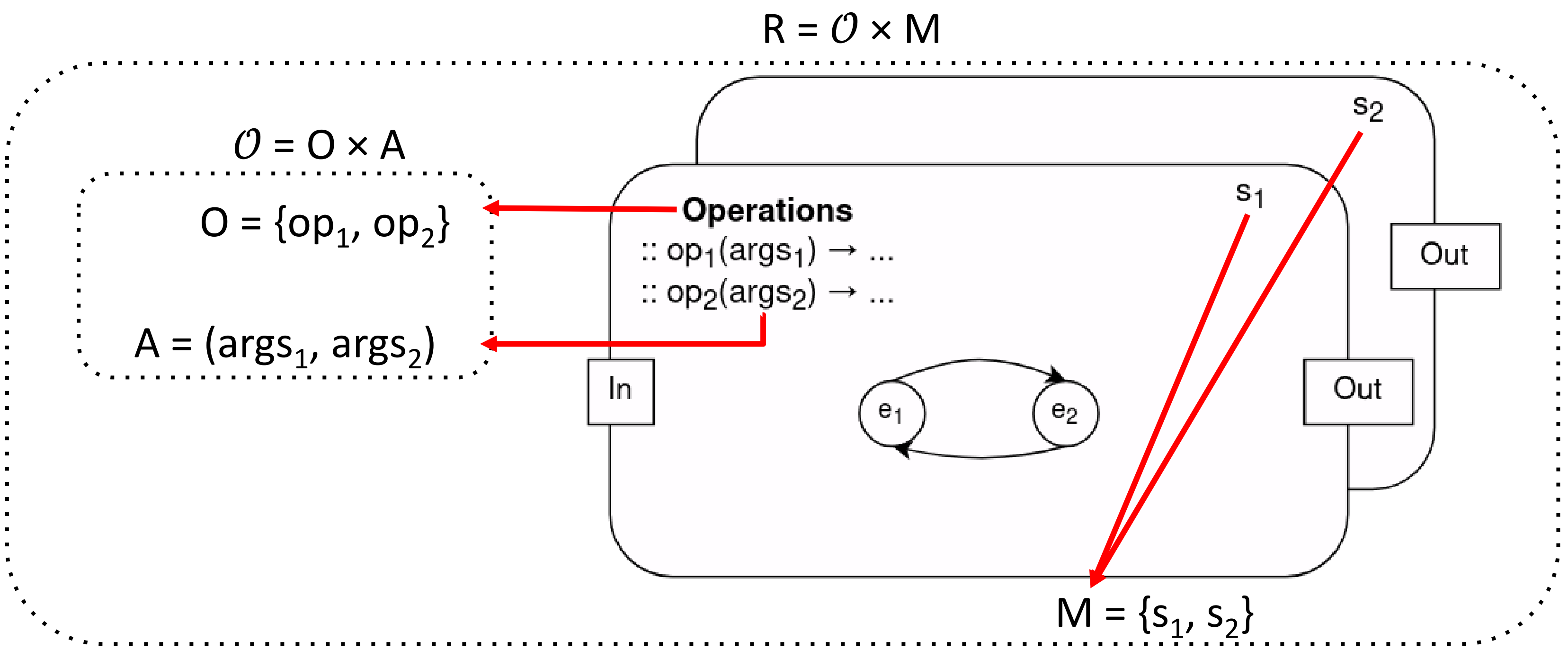
SMR Formalization: Definitions

- **Definition 2** - Operations with arguments
  - **Example: SMR 2 - Key-value store**

SMR 2 - Key-value store	**Arguments**	**Operations with arguments**
O = {get, put}	A = {a, b, ..., z}	$\mathcal{O}_{\text{get}}$ = {get} x A*
	A* set of all strings over A	(get, “firstkey”); (get, “secondkey”)

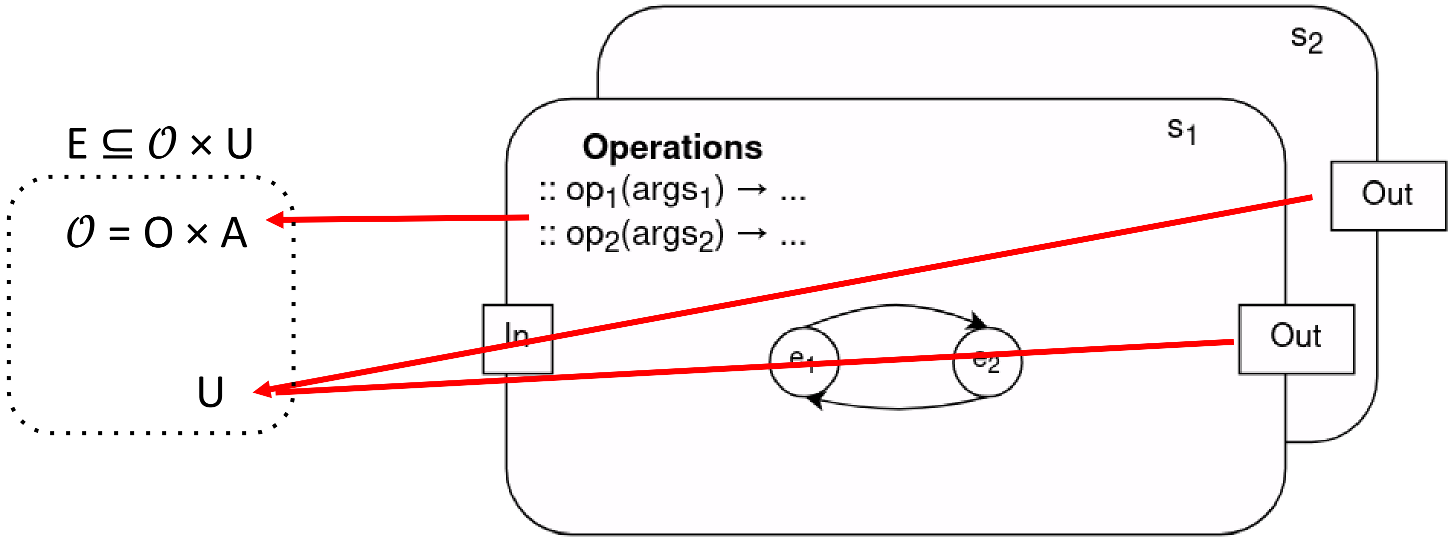
SMR Formalization: Definitions

- **Definition 3** - Replicated service with arguments


$R = \mathcal{O} \times M$
$\mathcal{O} = O \times A$
$O = \{op_1, op_2\}$
$A = (args_1, args_2)$
Operations
$:: op_1(args_1) \rightarrow ...$
$:: op_2(args_2) \rightarrow ...$
In
e_1
e_2
s_1
s_2
Out
Out
$M = \{s_1, s_2\}$

SMR Formalization: Definitions

- **Definition 4** – Execution
- **Definition 5** – Output



$E \subseteq \mathcal{O} \times U$
$\mathcal{O} = O \times A$
U
s_2
s_1
Operations
:: $op_1(args_1) \rightarrow$ ...
:: $op_2(args_2) \rightarrow$ ...
In
e_1
e_2
Out
Out

SMR Formalization: Example

SMR 2 - Key-value store

O = {get, put}

Arguments

A = {a, b, ..., z}

A* set of all strings over A

Operations with arguments

$\mathcal{O}_{get}$ = {get} x A*

(get, “firstkey”); (get, “secondkey”)

Execution and output

$E \subseteq \mathcal{O} \times U$

E ⊆ ({get} x A*) x A*

(get, “firstkey”) x (“firstvalue”)

Composing SMR

A new perspective

Composing State Machine Replication (CSMR)

Literature

Composition can be a powerful strategy

Existing SMR

Add new features combining different SMRs

Extending a service

Incorporating additional functionalities

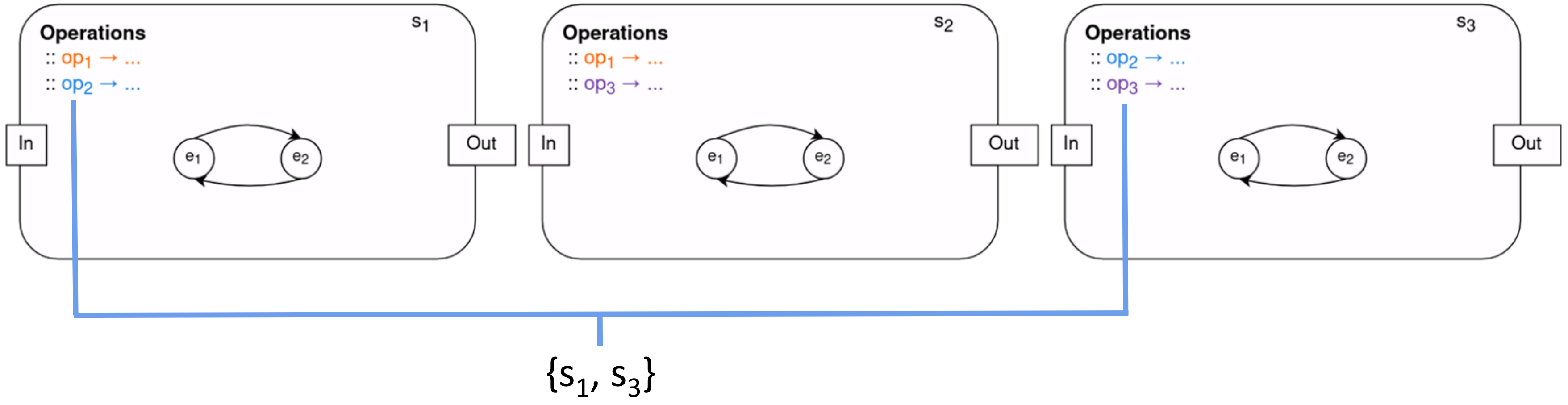
Different operations

Now there is no longer any requirement for all replicas to perform the same operations

CSMR: Definitions

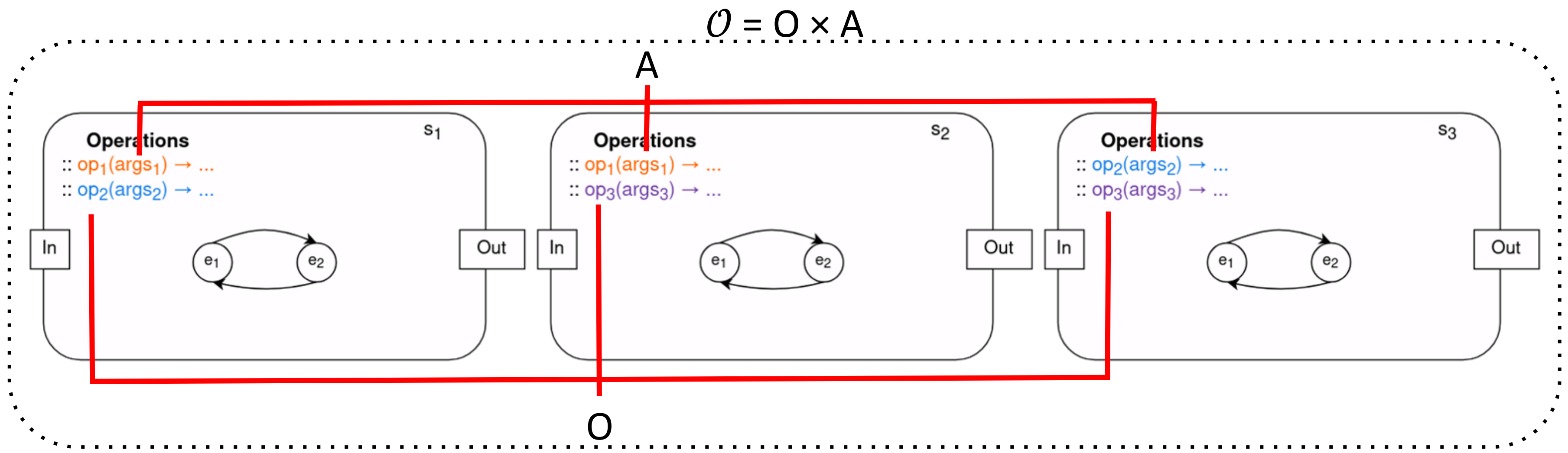
- **Definition 6** - Replication set

$$R(x) = \{s_i \in M : (x , s_i) \in R\}$$



CSMR: Definitions

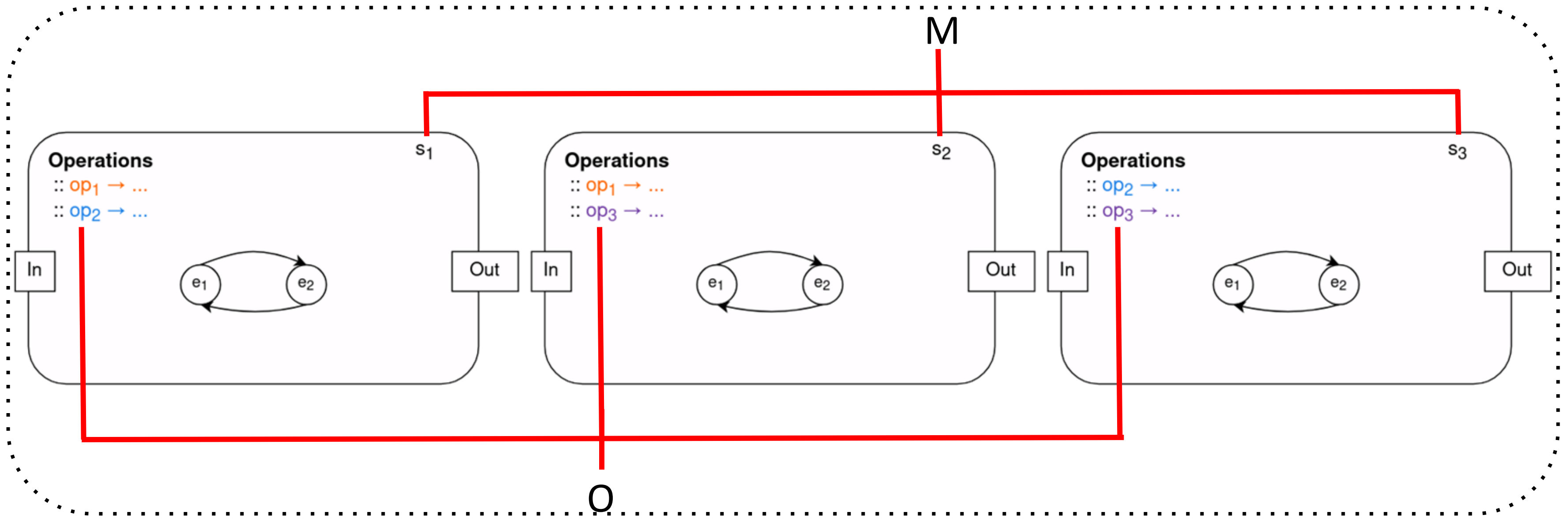
- **Definition 7** - Replication set with arguments



- $R(x) = \{s_i \in M : ((x, a), s_i) \in R, \text{ with } a \in A\}$

CSMR: Definitions

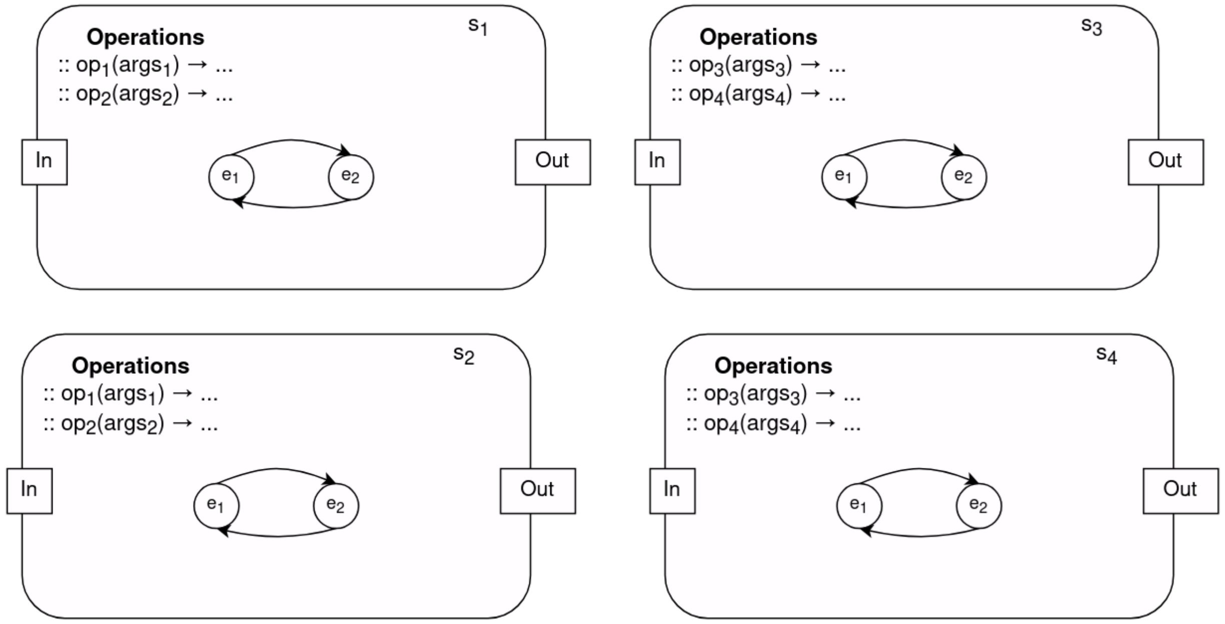
- **Definition 8** - Composable replicated service



$R \subseteq O \times M$

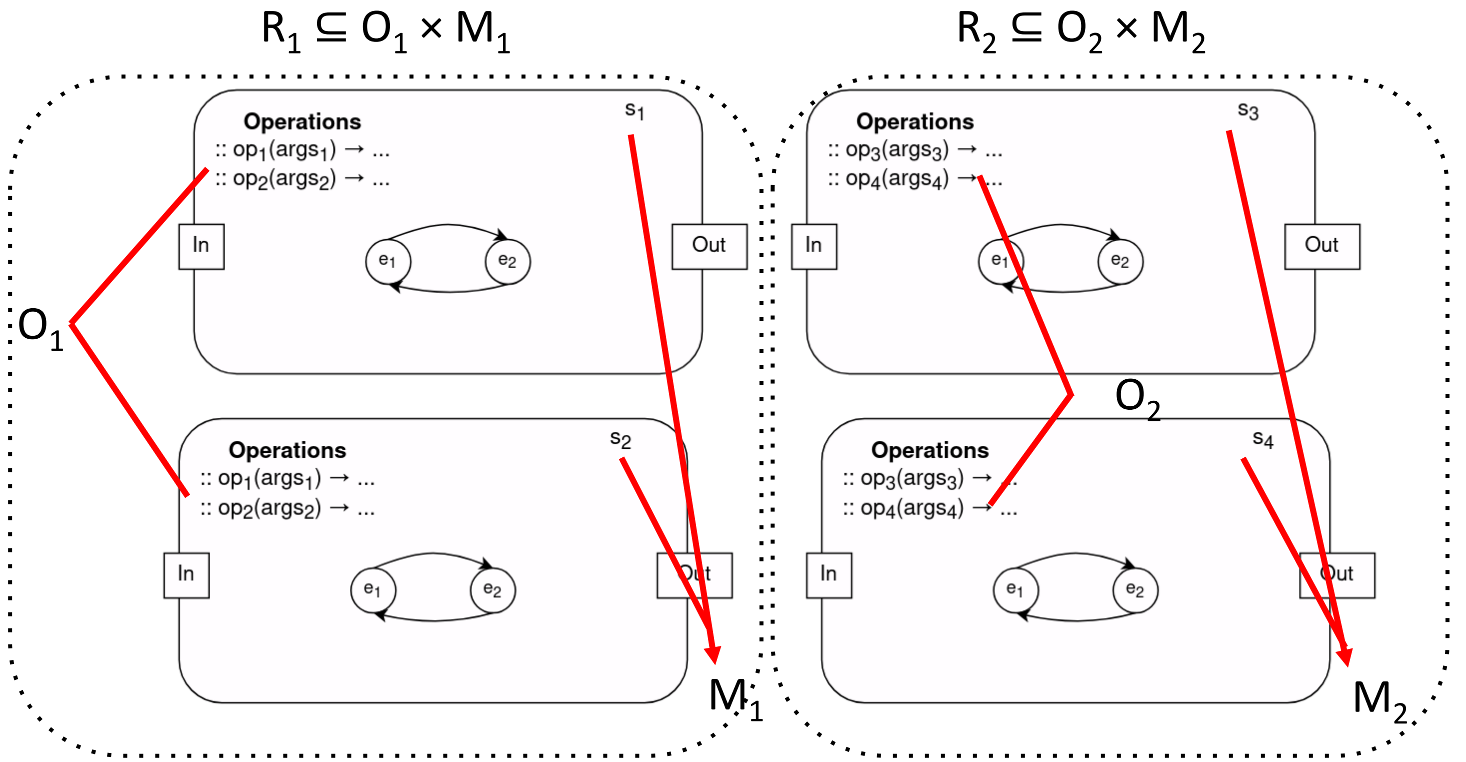
CSMR: Definitions

- **Definition 9** - Composition


Operations
:: op$_1$(args$_1$) → ...
:: op$_2$(args$_2$) → ...
s$_1$
In
Out
e$_1$
e$_2$
Operations
:: op$_3$(args$_3$) → ...
:: op$_4$(args$_4$) → ...
s$_3$
In
Out
e$_1$
e$_2$
Operations
:: op$_1$(args$_1$) → ...
:: op$_2$(args$_2$) → ...
s$_2$
In
Out
e$_1$
e$_2$
Operations
:: op$_3$(args$_3$) → ...
:: op$_4$(args$_4$) → ...
s$_4$
In
Out
e$_1$
e$_2$

CSMR: Definitions

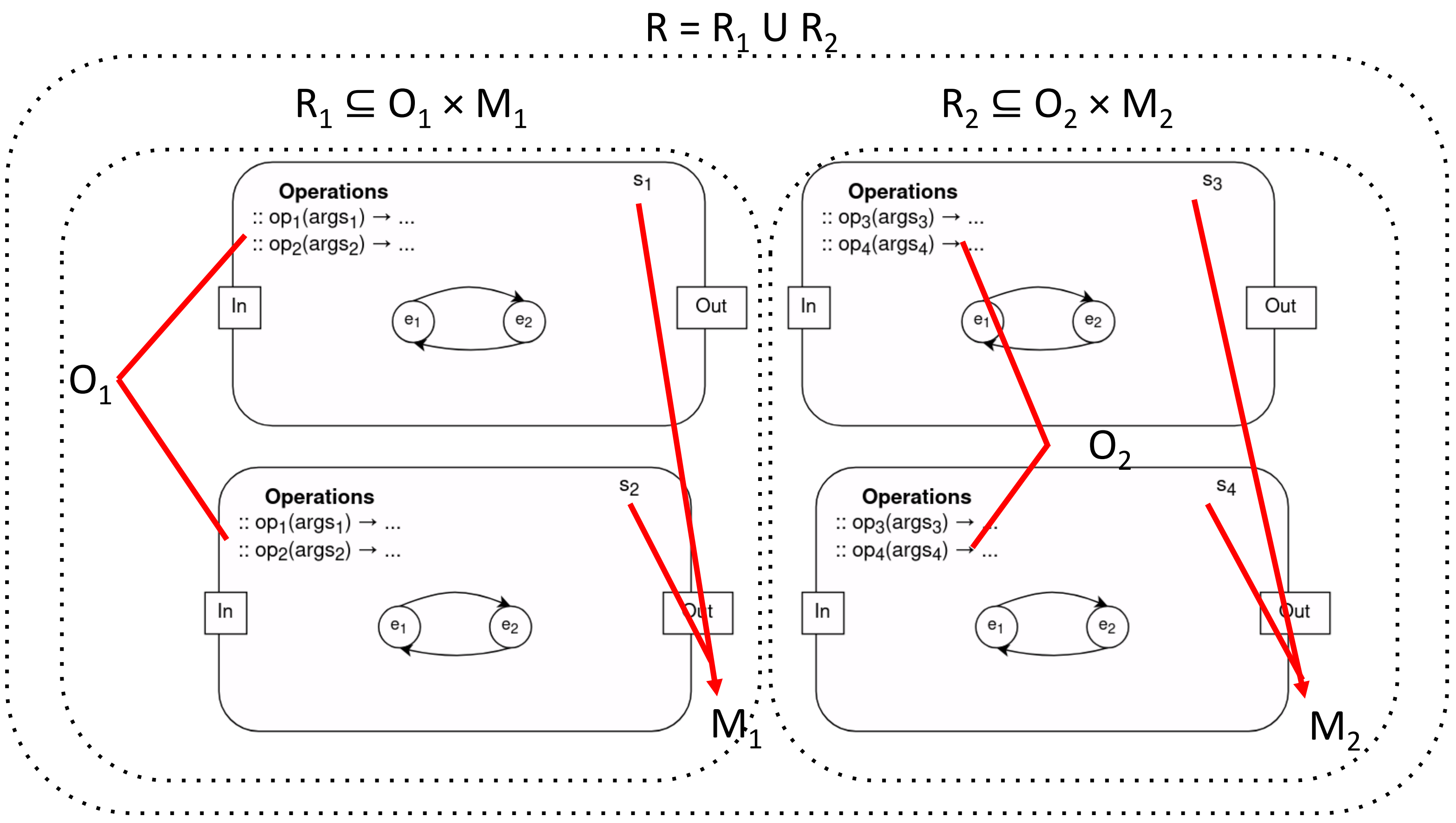
- **Definition 9** - Composition



$R_1 \subseteq O_1 \times M_1$
$R_2 \subseteq O_2 \times M_2$
Operations
:: $op_1(args_1) \to$ ...
:: $op_2(args_2) \to$ ...
s_1
In
Out
e_1
e_2
Operations
:: $op_1(args_1) \to$ ...
:: $op_2(args_2) \to$ ...
s_2
In
Out
e_1
e_2
O_1
M_1
Operations
:: $op_3(args_3) \to$ ...
:: $op_4(args_4) \to$ ...
s_3
In
Out
e_1
e_2
Operations
:: $op_3(args_3) \to$ ...
:: $op_4(args_4) \to$ ...
s_4
In
Out
e_1
e_2
O_2
M_2

CSMR: Definitions

- **Definition 9** - Composition

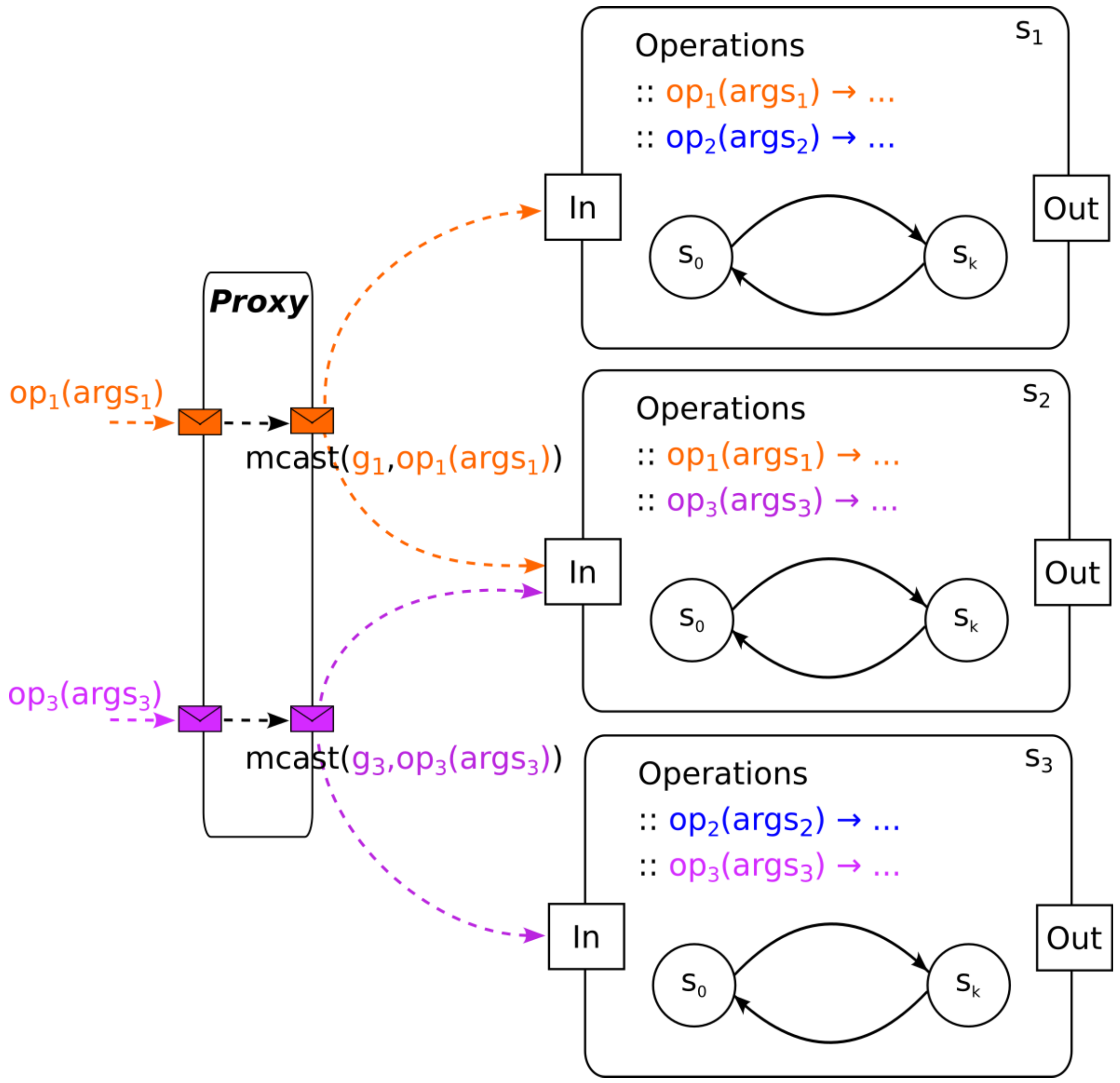


$R = R_1 \cup R_2$
$R_1 \subseteq O_1 \times M_1$
$R_2 \subseteq O_2 \times M_2$
Operations
:: $op_1(args_1) \rightarrow$ ...
:: $op_2(args_2) \rightarrow$ ...
:: $op_3(args_3) \rightarrow$ ...
:: $op_4(args_4) \rightarrow$ ...
s_1
s_2
s_3
s_4
In
Out
e_1
e_2
O_1
O_2
M_1
M_2

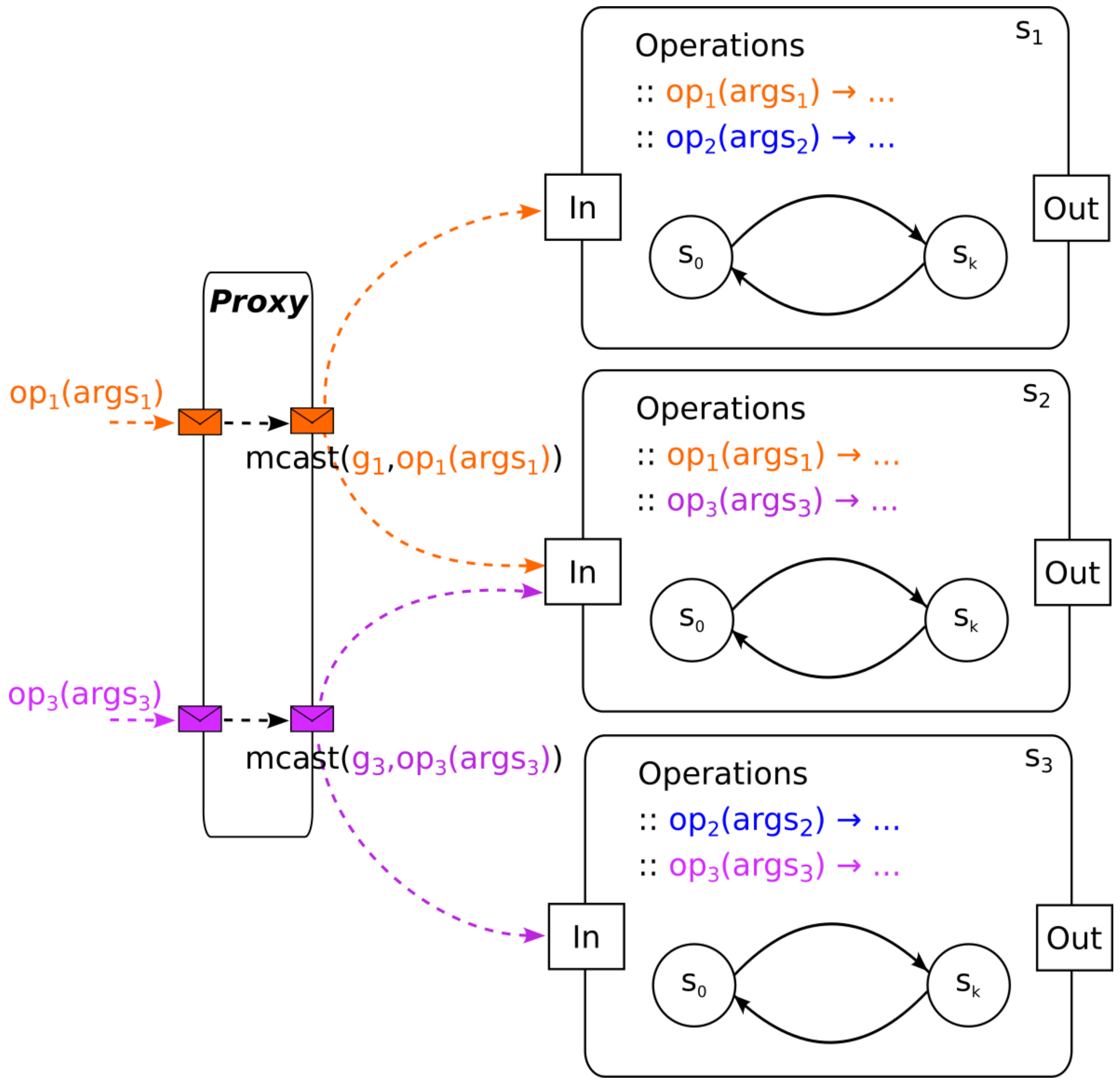
Composing strategies

- Adding SMR operations
- Extending SMR operations’ execution
- Argument partition

Composing strategies – Adding operations



Composing strategies – Adding operations


Proxy
$op_1(args_1)$
$mcast(g_1, op_1(args_1))$
$op_3(args_3)$
$mcast(g_3, op_3(args_3))$
s_1
Operations
$:: op_1(args_1) \rightarrow ...$
$:: op_2(args_2) \rightarrow ...$
In
Out
s_0
s_k
s_2
Operations
$:: op_1(args_1) \rightarrow ...$
$:: op_3(args_3) \rightarrow ...$
In
Out
s_0
s_k
s_3
Operations
$:: op_2(args_2) \rightarrow ...$
$:: op_3(args_3) \rightarrow ...$
In
Out
s_0
s_k

SMR 2 – Key-value store

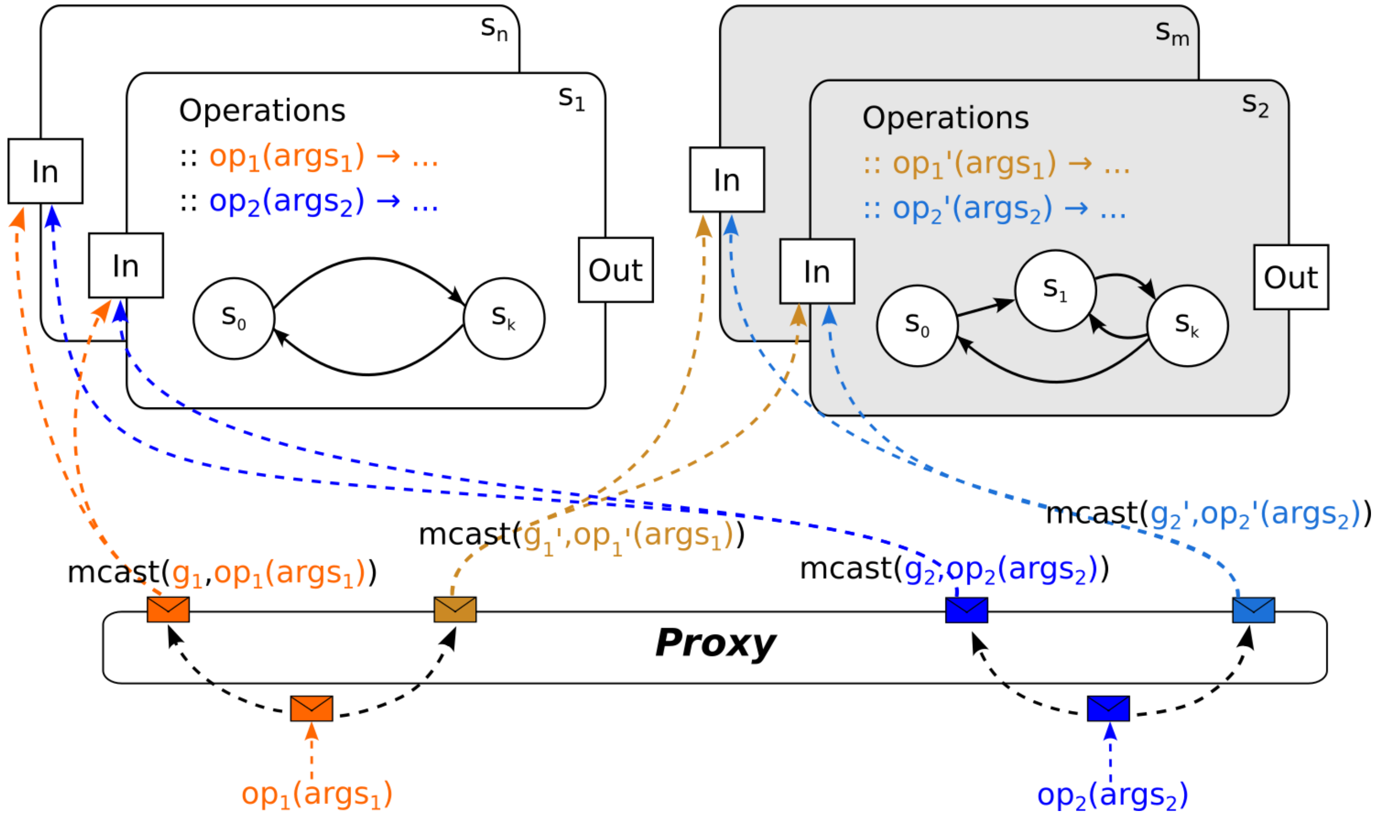
O = {get, put}

+

SMR 1 – Lock service

O = {acquire, release}

Composing strategies – Extending operations execution



s_n
s_1
Operations
:: $op_1(args_1) \rightarrow \ldots$
:: $op_2(args_2) \rightarrow \ldots$
In
In
Out
s_0
s_k
s_m
s_2
Operations
:: $op_1'(args_1) \rightarrow \ldots$
:: $op_2'(args_2) \rightarrow \ldots$
In
In
Out
s_0
s_1
s_k
mcast(g_2',$op_2'(args_2)$)
mcast(g_1',$op_1'(args_1)$)
mcast(g_2,$op_2(args_2)$)
mcast(g_1,$op_1(args_1)$)
Proxy
$op_1(args_1)$
$op_2(args_2)$

SMR 2 – Key-value store

O = {get, put}

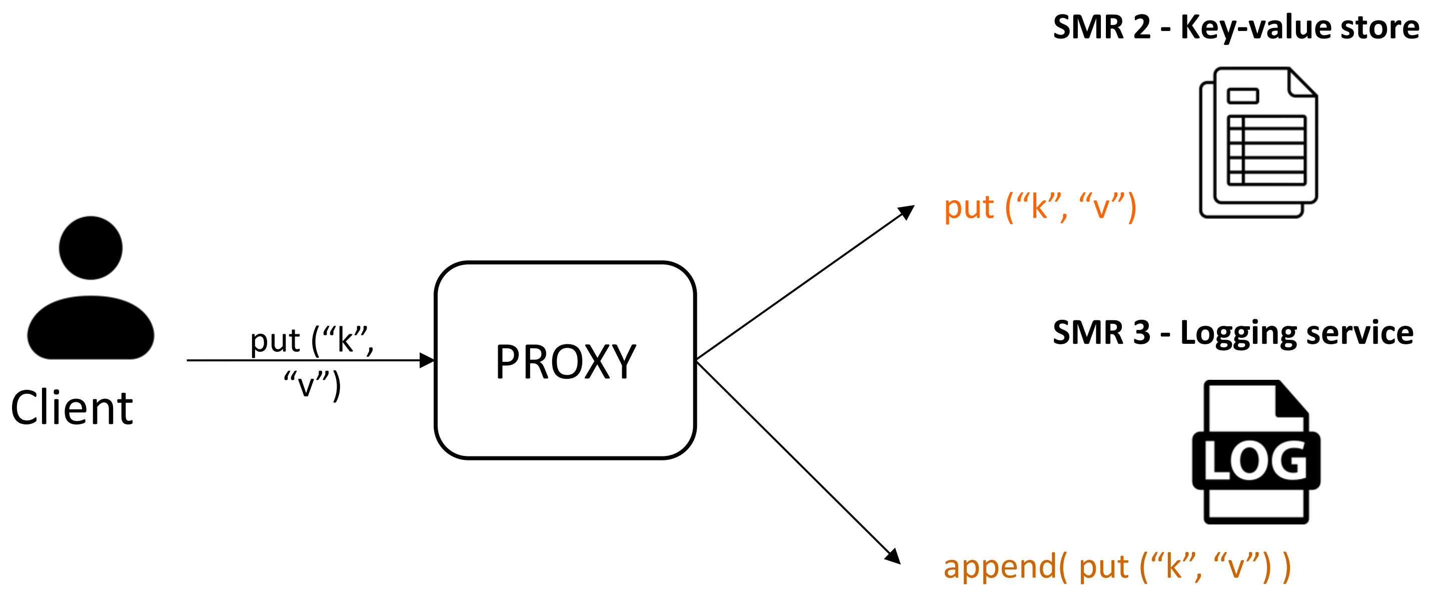
+

SMR 3 – Logging Service

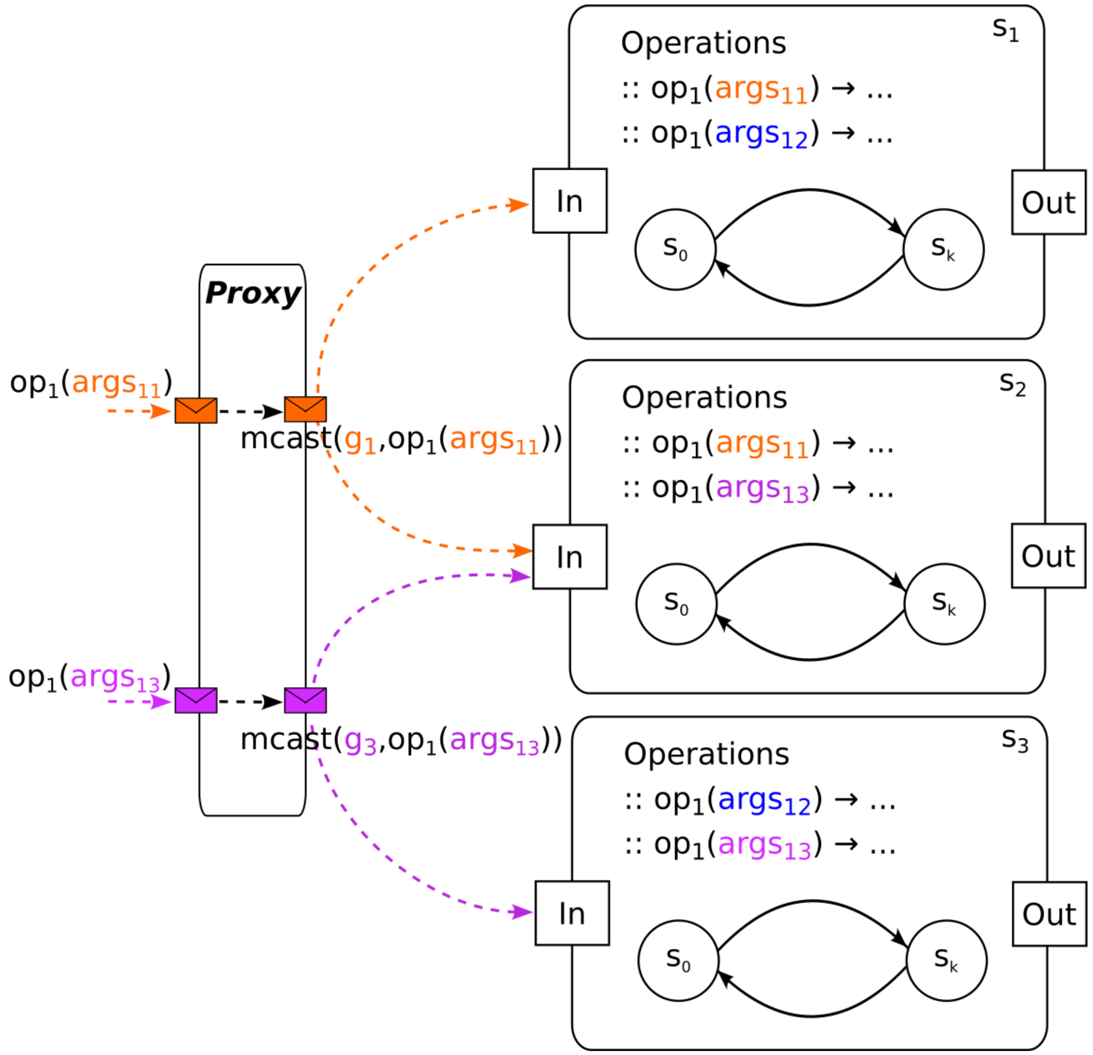
O = {append, retrieve, truncate}

Composing strategies – Extending operations execution

- Example: Key-value store with logging



Composing strategies – Argument partition



SMR 2 – Key-value store

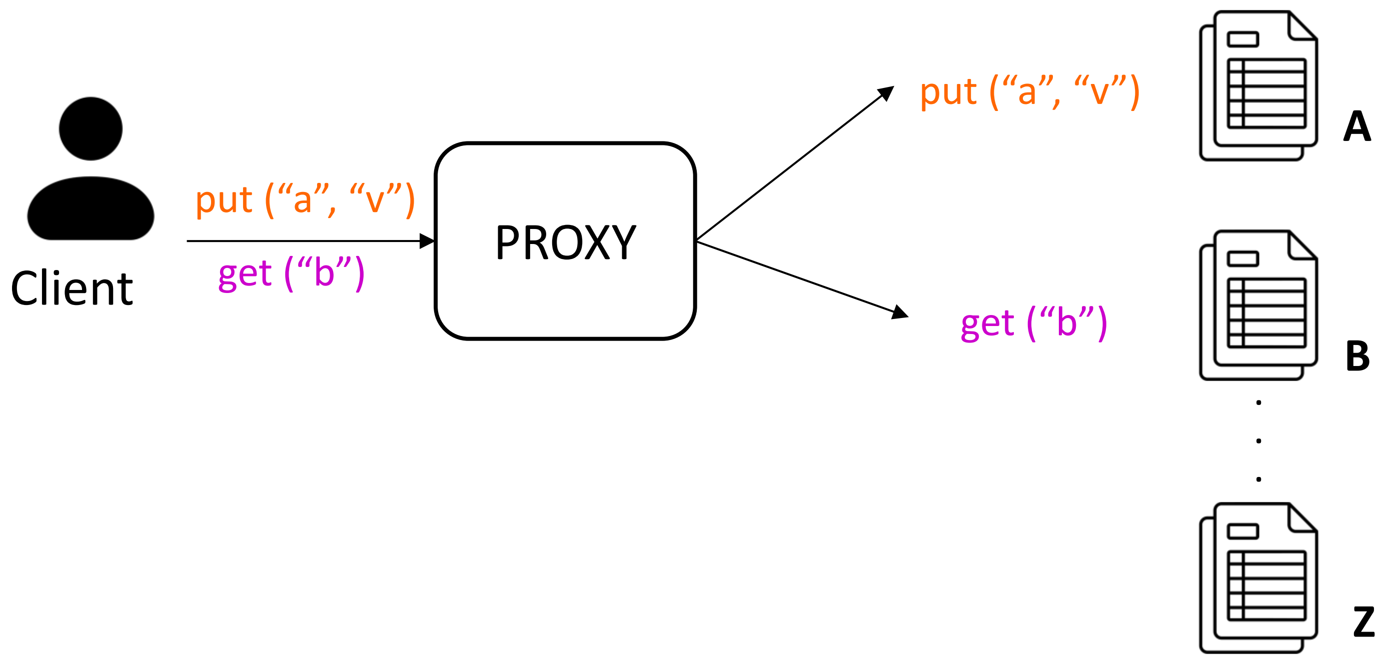
$O = \{get, put\}, A = \{A_a , A_a \times A\}$

+

SMR 2 – key-value store

$O = \{get, put\}, A = \{A_b , A_b \times A\}$

Composing strategies – Argument partition



Final remarks

- CSMR enables the construction of more complex applications
  - Still, preserving fault tolerance

- Modular approach encourages development of loosely coupled architectures
  - Fits well to microservices and cloud applications

- Some of the composing strategies resemble previous contributions in the literature
  - Bezerra et al. Scalable state-machine replication (DSN, 2014)
  - Xavier et al. Scalable and decoupled logging for state machine replication (SBRC, 2021)

Future work

- Define an RPC API for client requests invocation
- Define a declarative configuration for CSMR (YAML file)
- Implement the Proxy
    - Many challenges
- Propose new use cases
    - Do you have any ideas? ☺

THANK YOU

Composing State Machine Replicas

Contacts:

Odorico Machado Mendizabal
odorico.mendizabal@ufsc.br
https://www.inf.ufsc.br/~odorico.mendizabal/

Caroline Martins Alves
caroline.martins@posgrad.ufsc.br

Thaís Bardini Idalino
thais.bardini@ufsc.br
https://thaisidalino.github.io/