Privacy-Preserving Analytics in and out of the Clouds

Jon Crowcroft,

http://www.cl.cam.ac.uk/~jac22

1. Private Data Center->Public Cloud

- Health or Finance data, for example
  - Farr/NHS Scotland
  - HSBC
- Motives for public cloud
  - Scale out/cost save
  - Higher Throughput analytics
  - Share "access" with more researchers
  - <Yours goes here>

Infrastructure Location

- Keep friends&enemies near:
  - Legal/Regulatory Stuff (incl GDPR)
  - Latency/Availability etc
  - Control (physical access etc)
- Need to virtualise these (better)
  - Crypt Data at rest
  - Crypt data during “processing”
  - key management etc
  - ***Enclave***… SGX,Trust Zone, AMD, CHERI

GDPR – 2018 – right to an explanaion

- MISTAKES HAPPEN
  - ERROR ON INPUT
    - E.G.AMOUT OF $, AGE, ETC
  - MISTAKE IN CODE
    - E.G. IF (C=3) {} ...
  - MISTAKE IN TRAINING
    - E.G. SELECTION BIAS – PROB OF RE-OFFENDING ONLY TRAINED ON OFFENDERS
  - MISTAKE IN ML/INFERENCE
    - E.G. ACQUIRE A LATENT VARIABLE/RULE == GENDER (OR AGE)
  - SOMETHING WE HAVNT THOUGH OF YET
    - EMERGENCE?
- RIGHT TO REDRESS, AND BALANCE ASYMMETRIC POWER

MARU….@ turing.ac.uk

- ATI w/ Intel, Dstl, Docker, Microsoft

- Compare what is in SGX
  - Enter/leave cost, crypt memory o/h etc
  - Hypervisor?
- Compare w/ container on trustzone, cheri, AMD etc
  - Common APIs for keys etc
  - Virtualize?
- Pen test
  - many side channel pb
  - What if weak homomorphic crypto & diff priv?

Outline

1. Motivation: Trustworthy data processing in untrusted clouds

2. Overview of Intel SGX

3. Description of SGX-LKL Design

4. Description of preliminary SGX-Spark Design

5. Source code release of Java support on GitHub

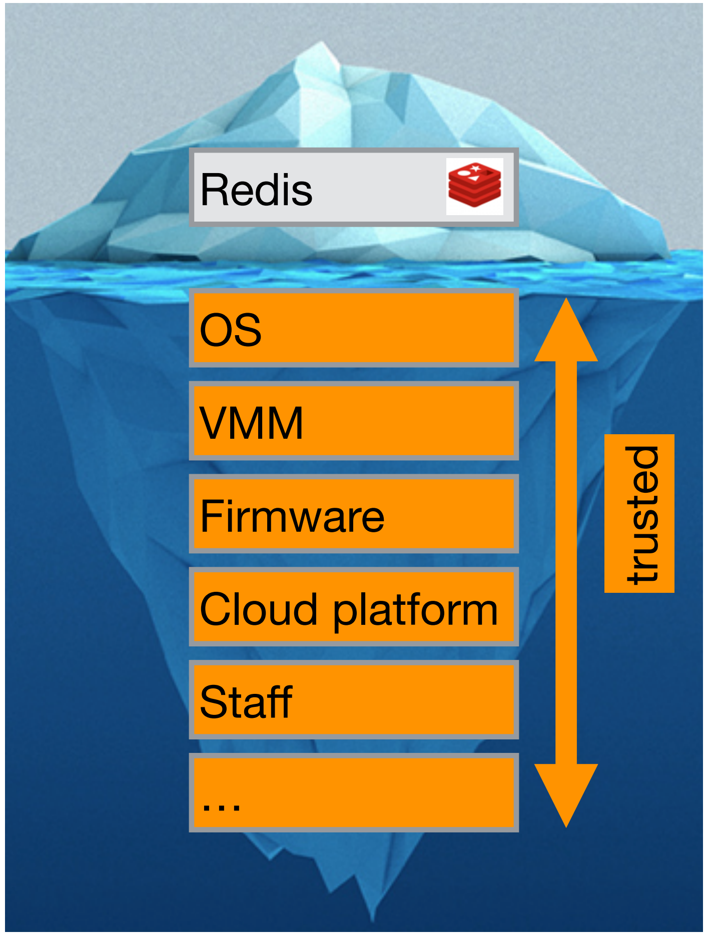
1. Motivation: Trustworthy Data Processing

Trust Issues: Provider Perspective

Cloud provider does not trust users

Use virtual machines to isolate users from each other and the host

VMs only provide one way protection

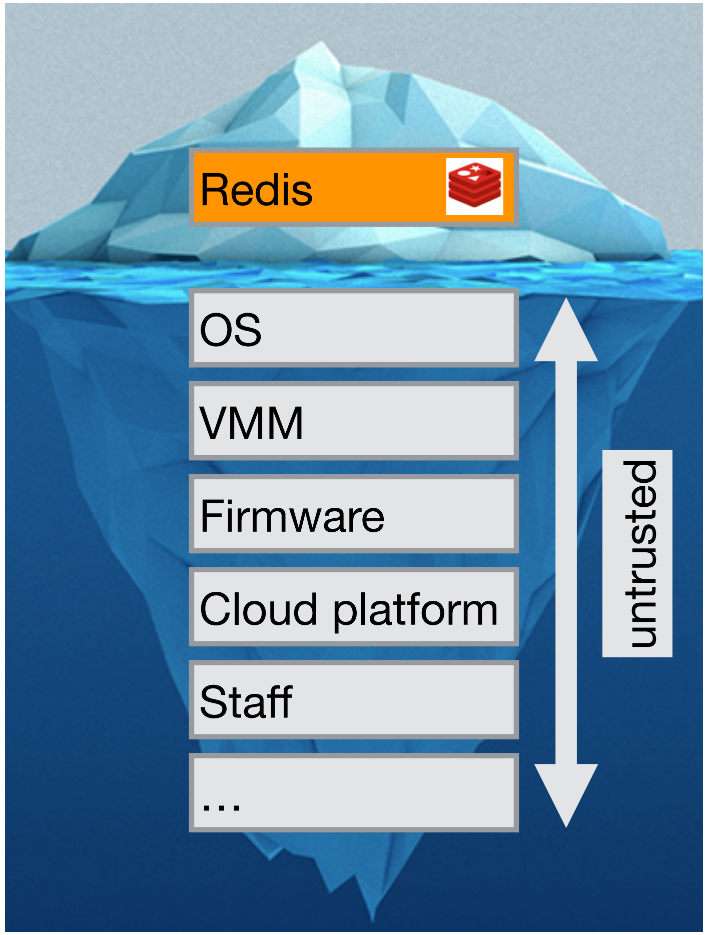


Trust Issues: User Perspective

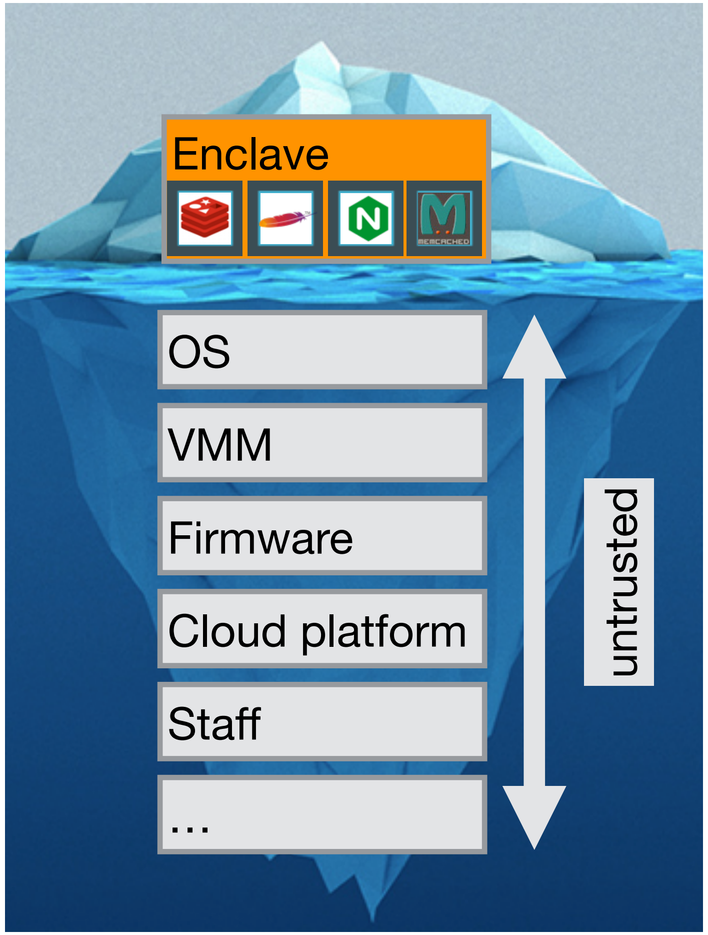
Users trust their applications

Users must implicitly trust cloud provider

Existing applications implicitly assume trusted operating system



Trusted Execution Support with Intel SGX



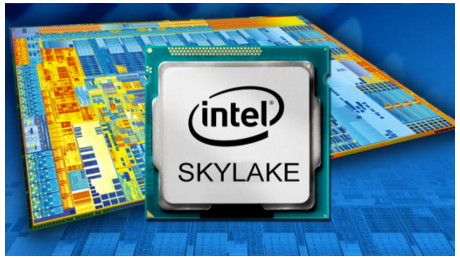
Users create HW-enforced trusted environment (enclave)

Supports unprivileged user code

Protects against strong attacker model

Remote attestation

Available on commodity CPUs

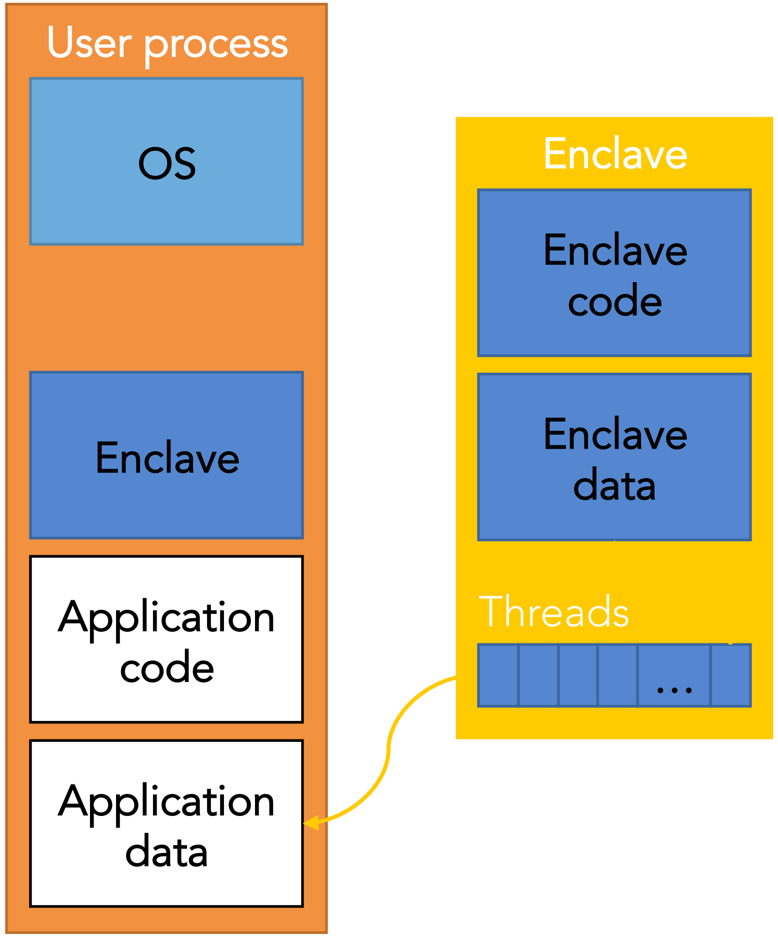


2. Overview of Intel SGX

Trusted Execution Environments

Trusted execution environment (TEE) in process

- Own code & data
- Controlled entry points
- Provides confidentiality & integrity
- Supports multiple threads
- Full access to application memory



Intel Software Guard Extensions (SGX)

Extension of Instruction Set Architecture (ISA) in recent Intel CPUs

- Skylake (2015), Kaby lake (2016)

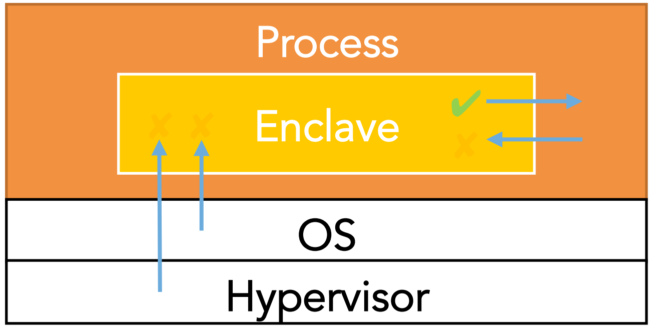
Protects confidentiality and integrity of code & data in untrusted environments

- Platform owner considered malicious
- Only CPU chip and isolated region trusted

SGX Enclaves

SGX introduces notion of **enclave**

- Isolated memory region for code & data
- New CPU instructions to manipulate enclaves and new enclave execution mode



Enclave memory **encrypted** and **integrity-protected** by hardware

- Memory encryption engine (MEE)
- No plaintext secrets in main memory

Enclave memory can be accessed only by enclave code

- Protection from privileged code (OS, hypervisor)

Application has ability to defend secrets

- Attack surface reduced to just enclaves and CPU
- Compromised software cannot steal application secrets

SGX SDK Code Sample

SGX application: untrusted code

```
char request_buf[BUFFER_SIZE];
char response_buf[BUFFER_SIZE];

int main()
{
  ...
  while(1)
  {
    receive(request_buf);
    ret = EENTER(request_buf, response_buf);
    if (ret < 0)
      fprintf(stderr, "Corrupted message\n");
    else
      send(response_buf);
  }
  ...
}
```

Enclave: trusted code

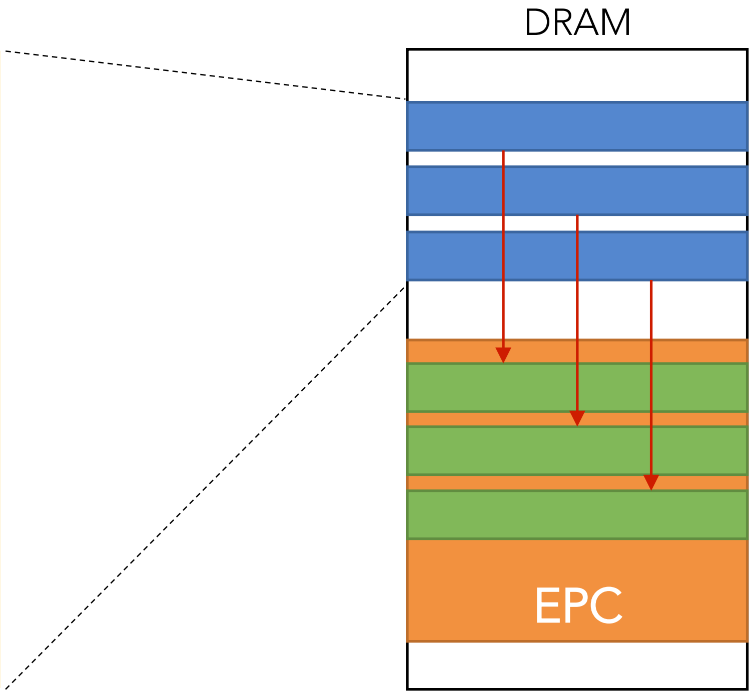
```
char input_buf[BUFFER_SIZE];
char output_buf[BUFFER_SIZE];

int process_request(char *in, char *out)
{
  copy_msg(in, input_buf);
  if(verify_MAC(input_buf))
  {
    decrypt_msg(input_buf);
    process_msg(input_buf, output_buf);
    encrypt_msg(output_buf);
    copy_msg(output_buf, out);
    EEXIT(0);
  } else
    EEXIT(-1);
}
```

Server:

- Receives encrypted requests
- Processes them in enclave
- Sends encrypted responses

SGX Enclave Construction

```
1 {  char input_buf[BUFFER_SIZE];
2 {  char output_buf[BUFFER_SIZE];

     int process_request(char *in, char *out)
     {
       copy_msg(in, input_buf);
       if(verify_MAC(input_buf))
       {
         decrypt_msg(input_buf);
3        process_msg(input_buf, output_buf);
         encrypt_msg(output_buf);
         copy_msg(output_buf, out);
         EEXIT(0);
       } else
         EEXIT(-1);
     }
```



Enclave populated using special instruction (EADD)

- Contents initially in untrusted memory
- Copied into EPC in 4KB pages

Both data & code copied before execution commences in enclave

SGX Enclave Construction

Enclave contents distributed in plaintext

- Must not contain any (plaintext) confidential data

Secrets provisioned after enclave constructed and integrity verified

Problem: what if someone tampers with enclave?

- Contents initially in untrusted memory

```
int process_request(char *in, char *out)
{
  copy_msg(in, input_buf);
  if(verify_MAC(input_buf))
  {
    decrypt_msg(input_buf);
    process_msg(input_buf, output_buf);
    encrypt_msg(output_buf);
    copy_msg(output_buf, out);
    EEXIT(0);
  } else
    EEXIT(-1);
}
```

```
int process_request(char *in, char *out)
{
  copy_msg(in, input_buf);
  if(verify_MAC(input_buf))
  {
    decrypt_msg(input_buf);
    process_msg(input_buf, output_buf);
    copy_msg(output_buf, external_buf);
    encrypt_msg(output_buf);
    copy_msg(output_buf, out);
    EEXIT(0);
  } else
    EEXIT(-1);
}
```

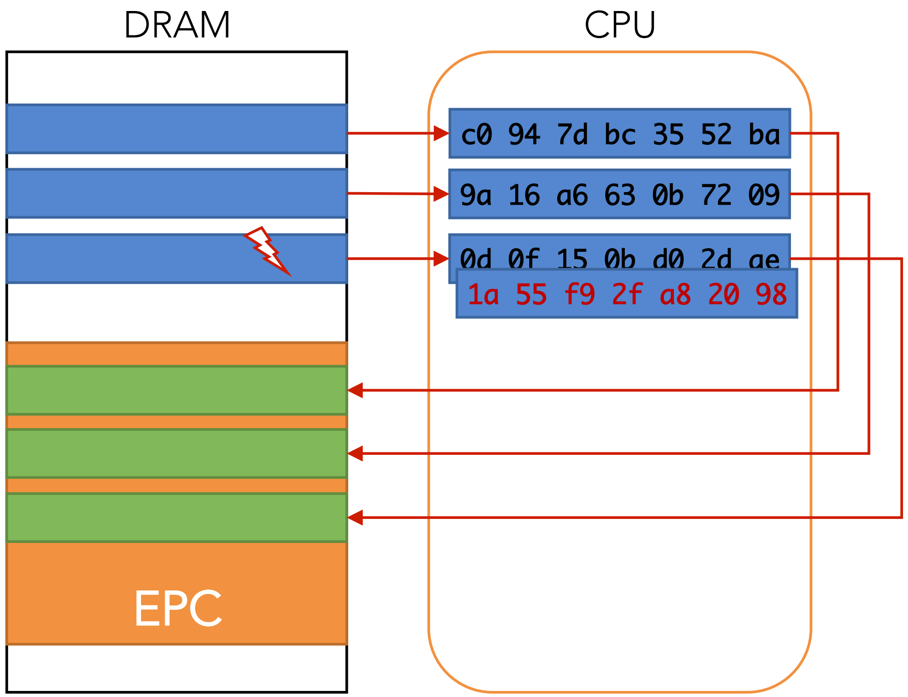
Write unencrypted response to outside memory

SGX Enclave Measurement

CPU calculates enclave measurement hash during enclave construction

- Each new page extends hash with page content and attributes (read/write/execute)
- Hash computed with SHA-256

Measurement can be used to attest enclave to local or remote entity



CPU calculates enclave measurement hash during enclave construction
Different measurement if enclave modified

SGX Enclave Attestation

Is my code running on remote machine intact?

Is code really running inside an SGX enclave?

Local attestation

– Prove enclave’s identity (= measurement) to another enclave on same CPU

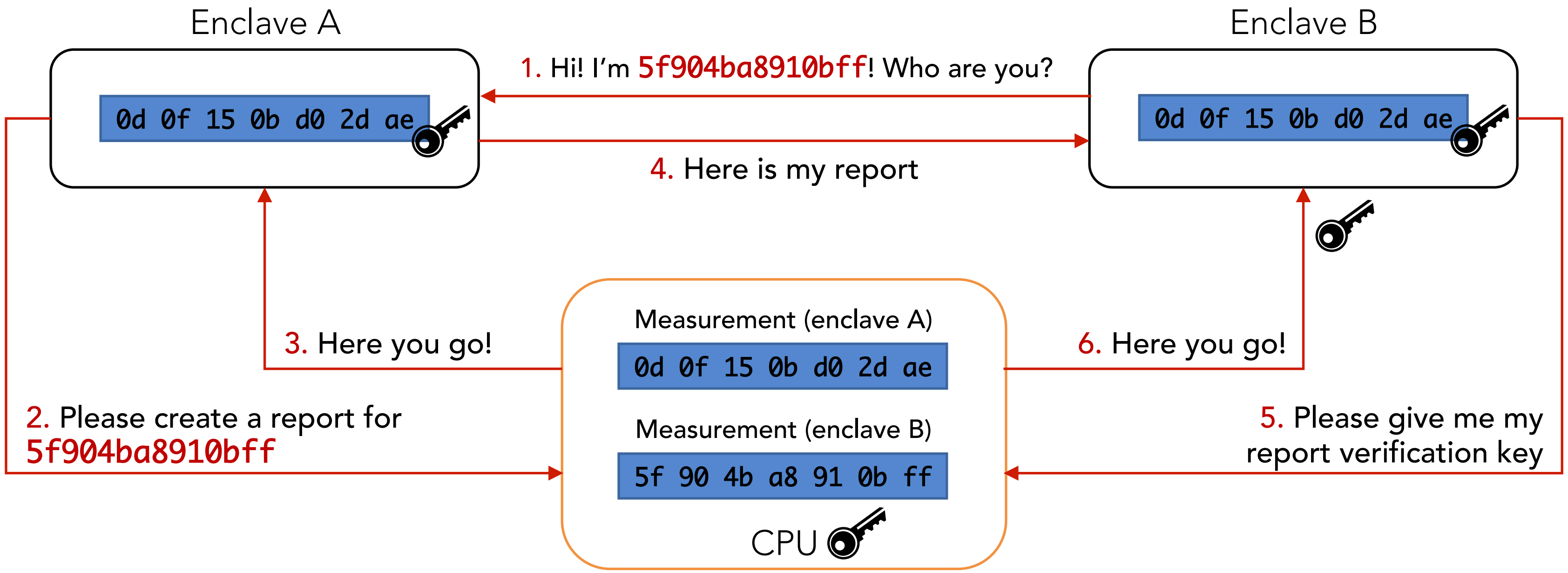
Remote attestation

– Prove enclave’s identity to remote party

Once attested, enclave can be trusted with secrets

Local Attestation

Prove identity of A to local enclave B



1. Target enclave B measurement required for key generation
2. Report contains information about target enclave B, including its measurement
3. CPU fills in report and creates MAC using report key, which depends on random CPU fuses and target enclave B measurement
4. Report sent back to target enclave B
5. Verify report by CPU to check that generated on same platform, i.e. MAC created with same report key (available only on same CPU)
6. Check MAC received with report and do not trust A upon mismatch

Remote Attestation

Transform local report to remotely verifiable “quote”

Based on provisioning enclave (PE) and quoting enclave (QE)

- Architectural enclaves provided by Intel
- Execute locally on user platform

Each SGX-enabled CPU has unique key fused during manufacturing

- Intel maintains database of keys

Remote Attestation

PE communicates with Intel attestation service

- Proves it has key installed by Intel
- Receives asymmetric attestation key

QE performs local attestation for enclave

- QE verifies report and signs it using attestation key
- Creates quote that can be verified outside platform

Quote and signature sent to remote attester, which communicates with Intel attestation service to verify quote validity

SGX Limitations & Research Challenges

Amount of memory enclave can use needs to be known in advance

- Dynamic memory support in SGX v2

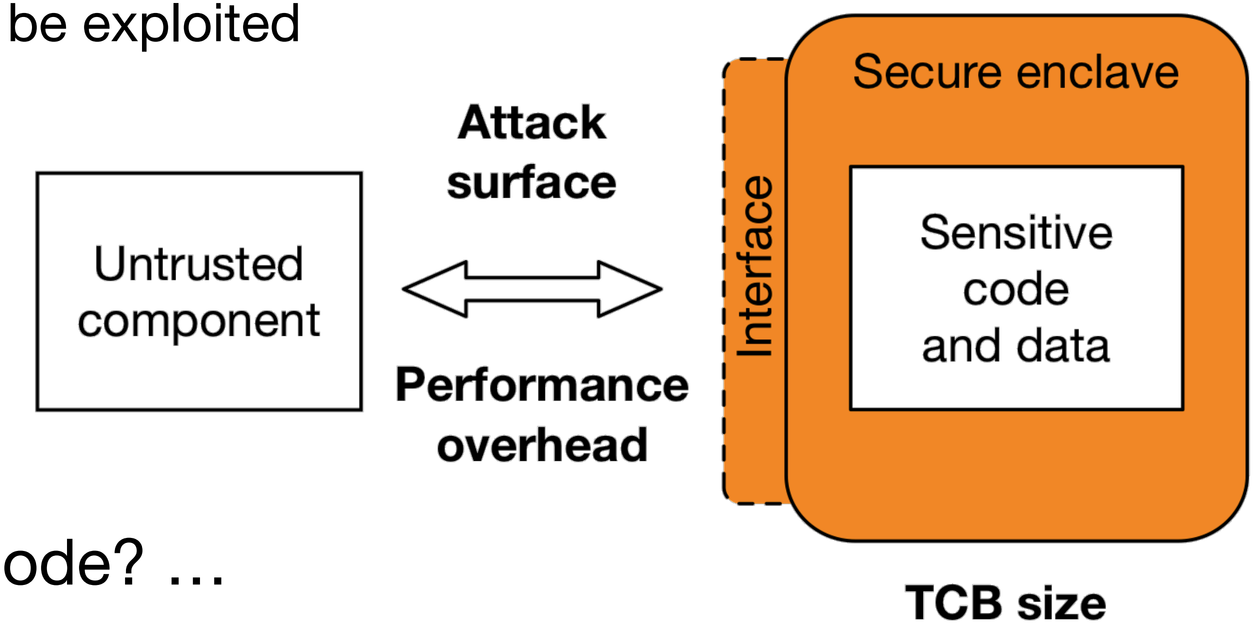
Security guarantees not perfect

- Vulnerabilities within enclave can still be exploited
- Side-channel attacks possible

Performance overhead

- Enclave entry/exit costly
- Paging very expensive

Application partitioning? Legacy code? …



3. Description of SGX-LKL

SGX-LKL: Supporting Managed Runtimes in SGX

Many applications need runtime support

- JVM
- .NET
- JavaScript/V8/Node.js

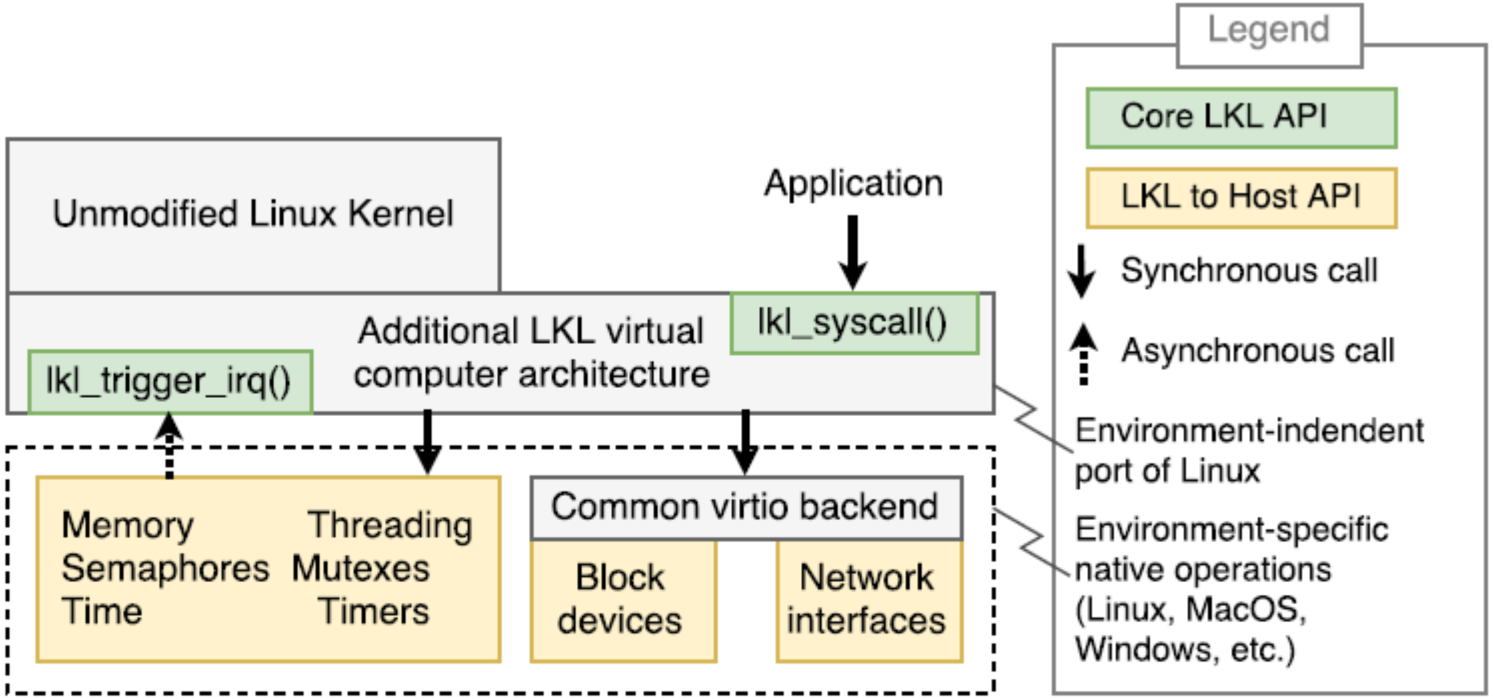
Requires complex system support

- Dynamic library loading
- Filesystem support
- Signal handling
- Complete networking stack

SGX-LKL: Linux Kernel Library inside SGX Enclaves

Based on **Linux Kernel Library (LKL)**

- Implemented as architecture-specific port of mainline Linux (`github.com/lkl`)
- Follows Linux no MMU architecture
- Full filesystem support
- Full network stack available



SGX-LKL Architecture

Runs **unmodified Linux applications** in SGX enclaves

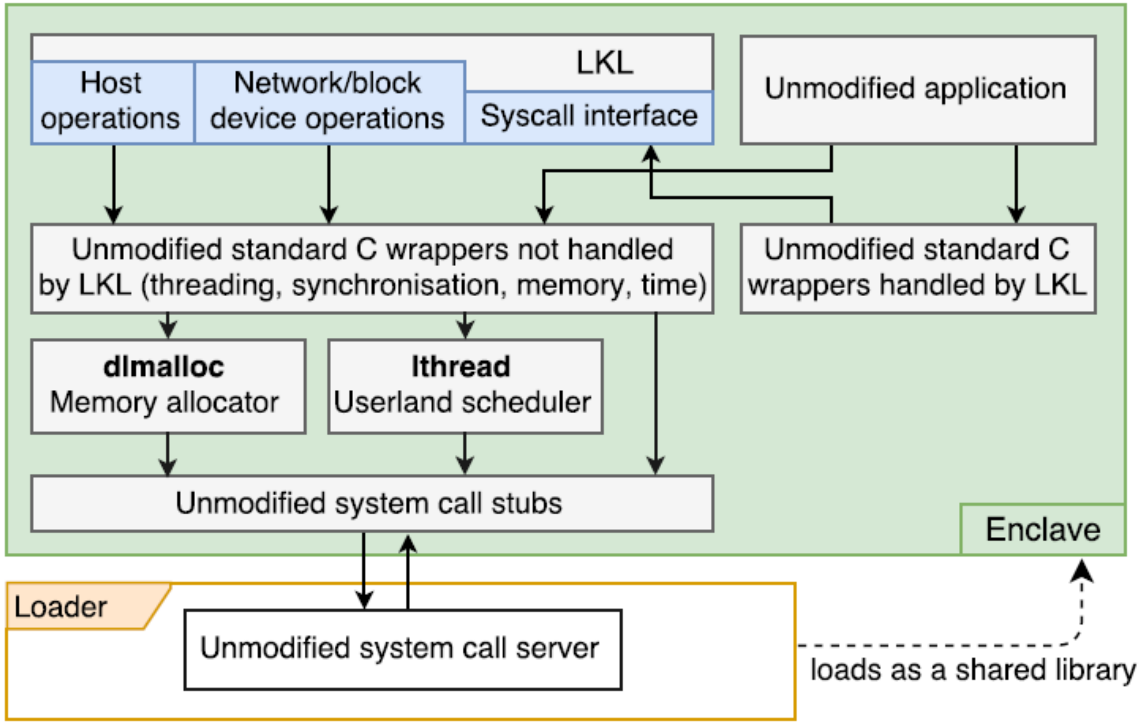
Applications and dependencies provided via **disk image**

Full Linux kernel functionality available

Custom memory allocator

User-level threading

- In-enclave synchronisation primitives


LKL
Host operations
Network/block device operations
Syscall interface
Unmodified application
Unmodified standard C wrappers not handled by LKL (threading, synchronisation, memory, time)
Unmodified standard C wrappers handled by LKL
dlmalloc Memory allocator
lthread Userland scheduler
Unmodified system call stubs
Enclave
Loader
Unmodified system call server
loads as a shared library

4. Description of SGX-Spark

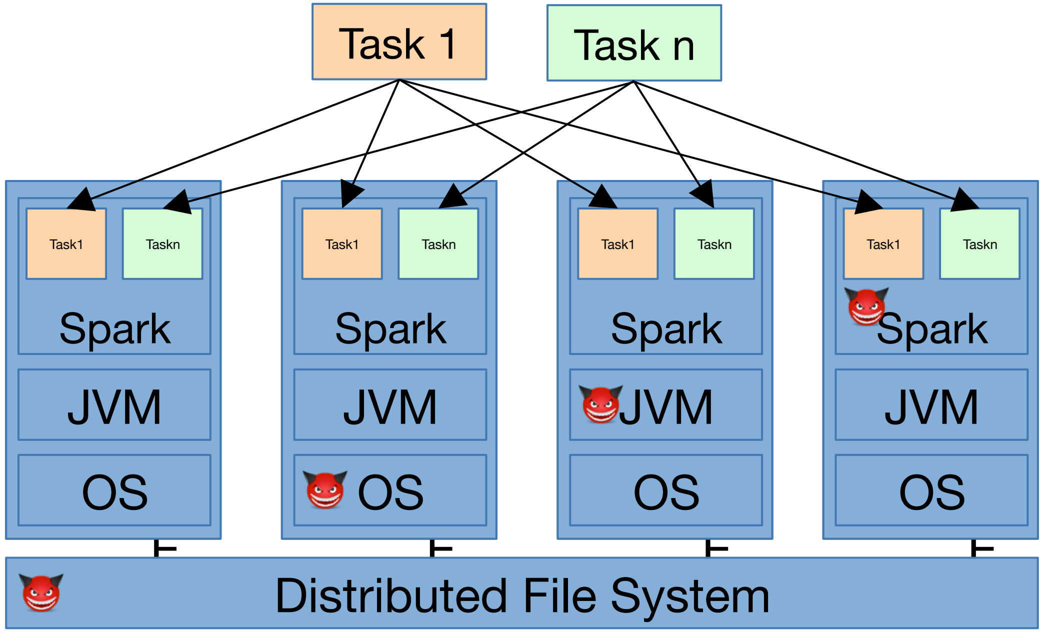
Secure Big Data Processing

Processing of large amounts of sensitive information

Outsourcing of data storage and processing

Cloud provider can access processed data

– Not acceptable for number of industries



```
def main(args: Array[String]) {
  new SparkContext(new SparkConf())
    .textFile(args(0))
    .flatMap(line => {line.split(" ")})
    .map(word => {(word, 1)})
    .reduceByKey{case (x, y) => x + y}
    .saveAsTextFile(args(1))
}
```

Secure Machine Learning

Secure **machine learning (ML)** killer application for Maru

- Resource-intensive thus good use case for cloud usage
- Raw training data comes with security impliations

Complex implementations of ML algorithms cannot be adapted for SGX

- Consider Spark MLlib with 100s of algorithms

Challenges

- Extremely **data-intensive** domain
- Must support **existing frameworks** (Spark, TensorFlow, MXNet, CNTK, …)
- ML requires **accelerators** support (GPUs, TPUs, …)
- Prevention of **side-channel** attacks

State of the Art

Protect confidentiality and integrity of tasks and input/output data

Opaque [Zheng, NSDI 2017]

- Hide access patterns of distributed data analytics (Spark SQL)
- Introduces new oblivious relational operators
- Does not support arbitrary/existing Scala Spark jobs

VC3 [Schuster, S&P 2015]

- Protects MapReduce Hadoop jobs
- Confidentiality/integrity of code/data; correctness/completeness of results
- No support for existing jobs → Re-implement for VC3

SGX Support for Spark

SGX-Spark

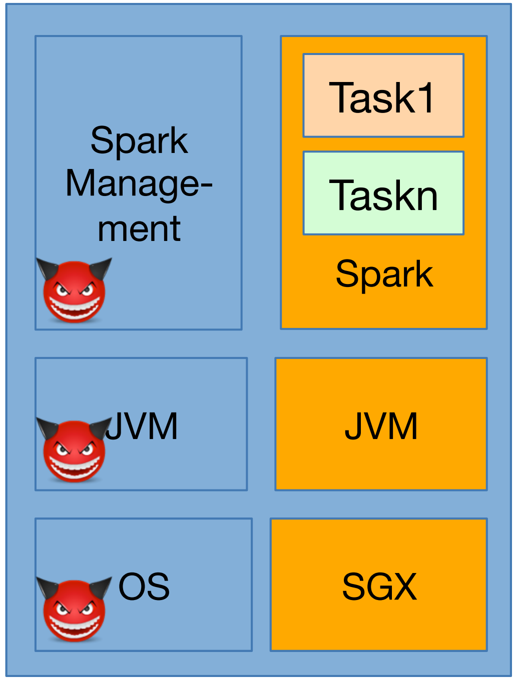
- Protect data processing from infrastructure provider
- Protect confidentiality & integrity of existing jobs
- No modifications for end users
- Acceptable performance overhead

Idea:
Execute only sensitive parts of Spark inside enclave

- Code that accesses/processes sensitive data

Code outside of enclave only accesses encrypted data

- Partition Spark
- Run two collaborating JVMs, inside enclave and outside of enclave



Challenges & Current State

1. **Partitioning Spark**

2. **Data movement between JVMs**

3. **Memory efficiency**

1. Partitioning Spark

Goal: Move minimal amount of Spark code to enclave

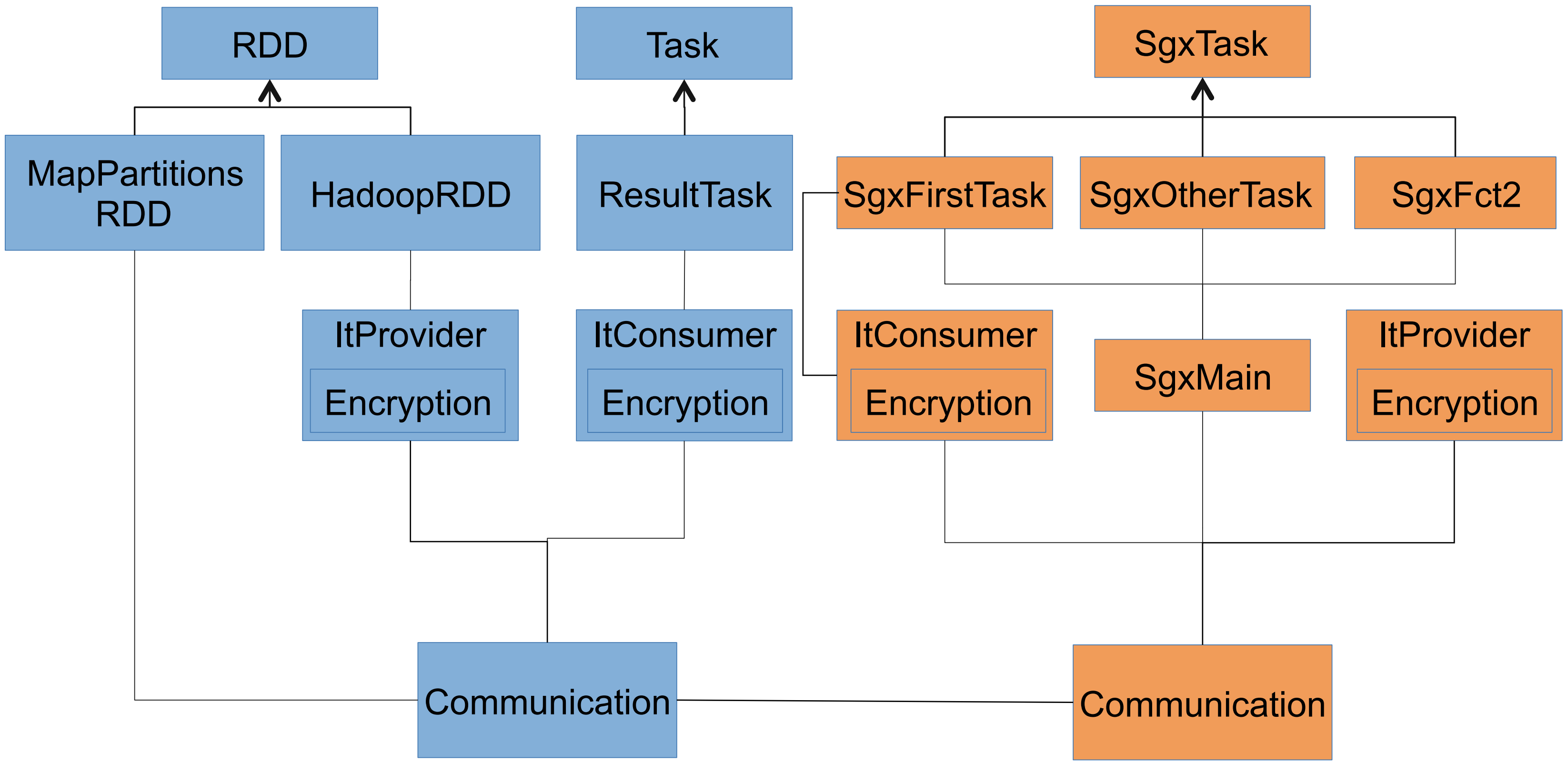
Outside	Enclave
HadoopRDD Provide iterator over input data partition (encrypted)	

`it`
MapPartitionsRDD
Execute user-provided function (f)
(eg `flatMap(line => {line.split(" ")})`)
(i) Serialise user-provided function `f`
`f,it`
(ii) Send `f` and `it` to enclave JVM
(iv) Receive result iterator `it_result`
`it2 = it_result`
	(iii) Decrypt input data
	(iv) Compute `f(it) = it_result`
	(v) Encrypt result

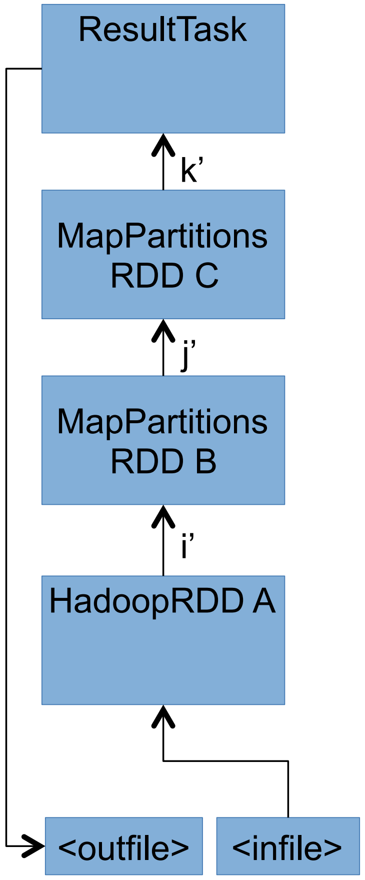
ExternalSorter
`g,it2`
Execute user-provided reduce function g
(eg `reduceByKey{case (x, y) => x + y}`)
`it2_result`
	(iii) Decrypt input data
	(iv) Compute `g(it2) = it2_result`
	(v) Encrypt result

ResultTask Output results	

1. Partitioning Spark

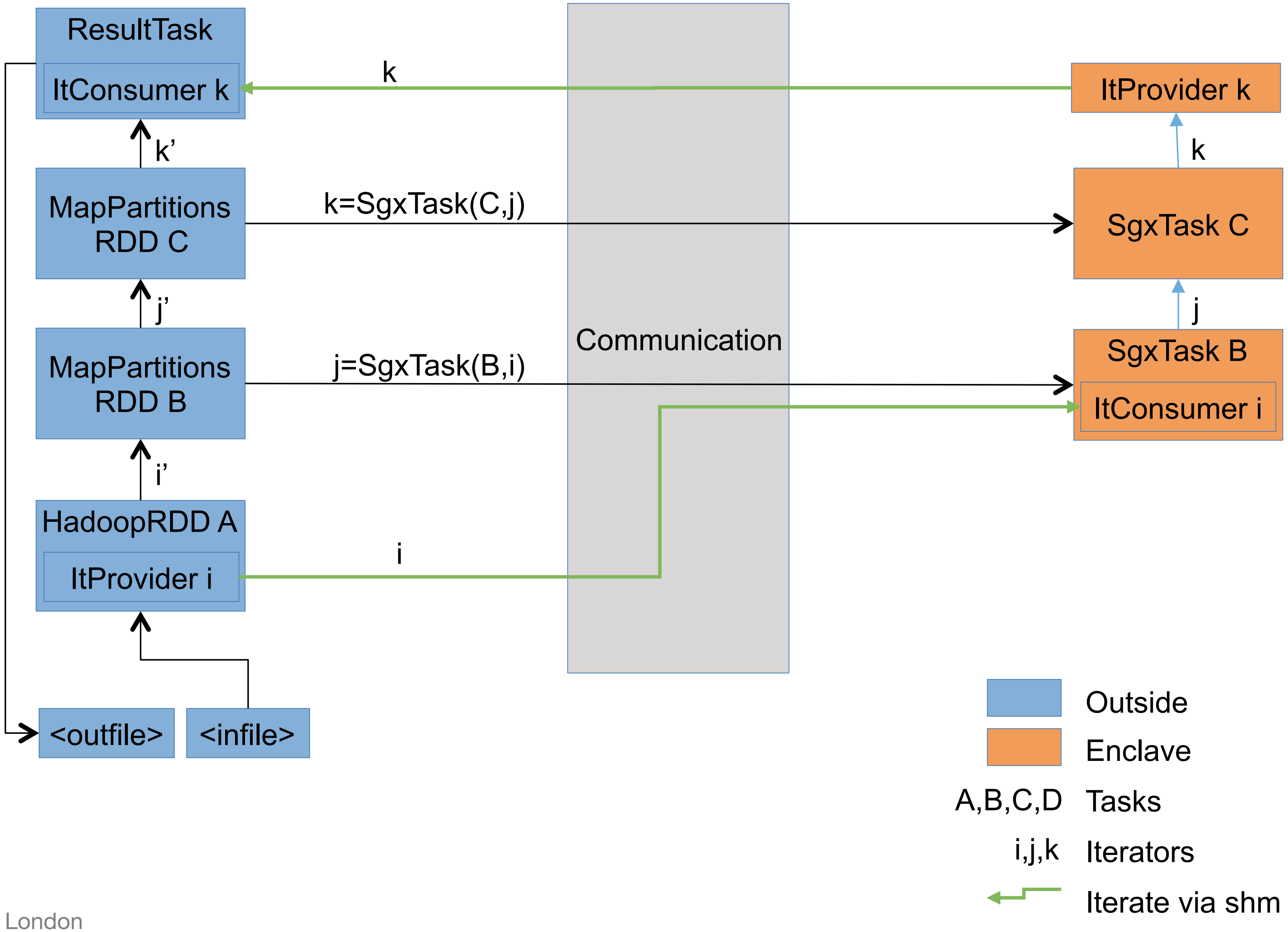


1. Partitioning Spark



🟦	Outside
🟧	Enclave
A,B,C,D	Tasks
i,j,k	Iterators
←	Iterate via shm

1. Partitioning Spark



2. Data Movement between JVMs

Goal: **Shared memory**

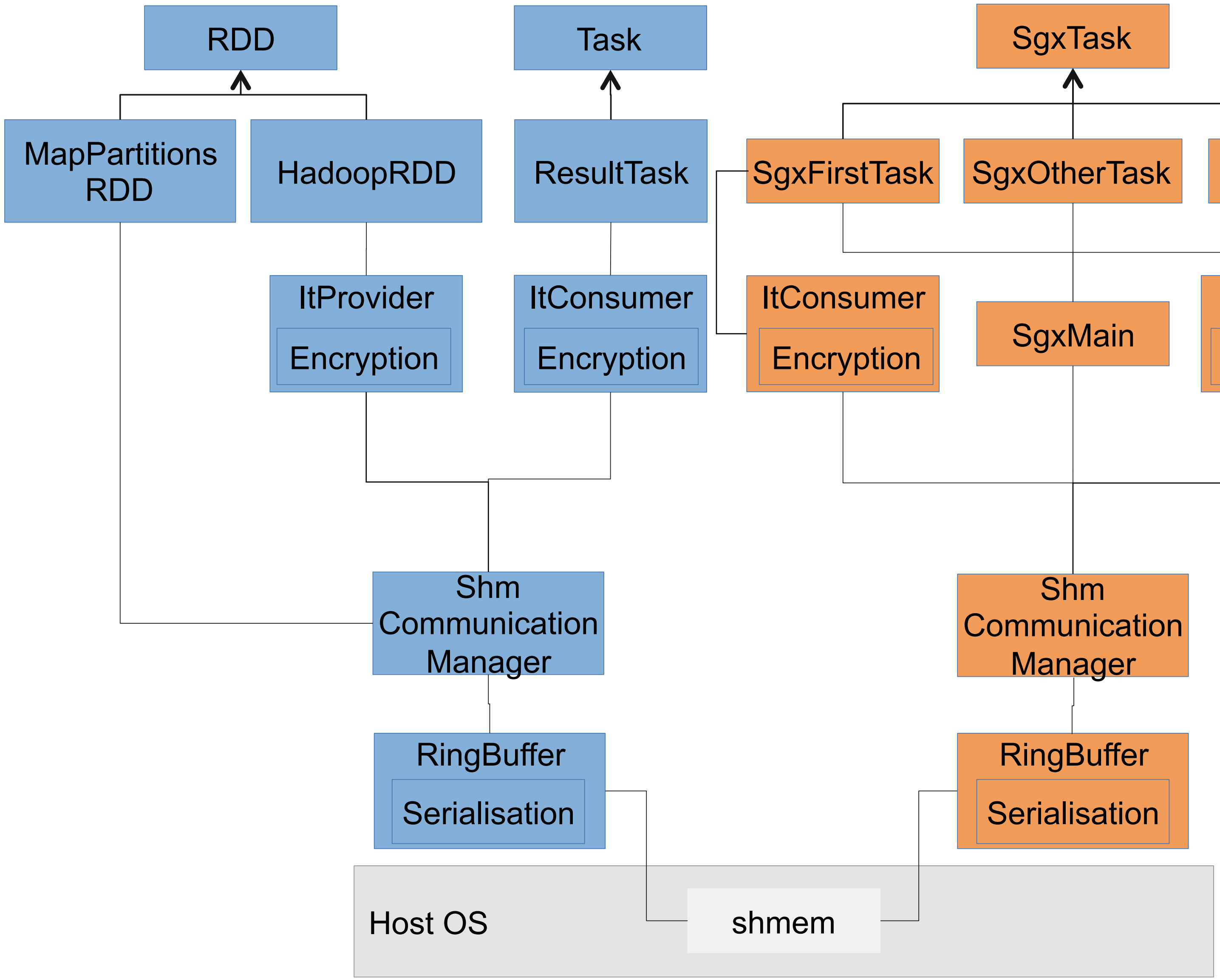
Use use host OS shared memory between two JVMs

- Outside access by enclave JVM

Manage shared memory between outside and enclave

Implement high-level read/write primitives

1. Partitioning Spark



2. Distributed Analytics

- Motives e.g.
    - Move code to data
    - Keep data close to owner/primary user
    - Guarantee can audit trail access
    - Add yours here
- Challenges
    - Depends on ML technology of choice & goal
        - PCA/Clustering, random forests
        - Curve fittign (regression etc)
        - Model Inferencing – e.g. Bayesian inference
    - Distrubuted differential privacy tricky
    - Hierarchical versus P2P?

Future Proof for GDPR

- Privacy by Design and by Default – HAT address all GDPR privacy requirement from its design principle to its security solution.
    - HAT ecosystem data exchange is based on fully specified privacy terms - time specific, recipient specific, minimum data points **specific** with **full intention disclosed.** Violation against any of such terms may result a ban from the Ecosystem.
- Consent by design and by default -
    - the PCST PoC mandates a "specific, informed and freely given and unambiguous" intension disclosure of data usage, for every single personal data access instances.
    - HAT technology ensures that an exchange is only authorised and kept valid by individual's case specific consent
- Rights for Individuals by design and by default – encapsulated personal data containers isolated for each individual, allows an individual is in full control of its HAT, hence inherently owns all of the following:
    - Right to Access | Right to be informed | Right to rectification | Right to restrict processing | Right to object to market
    - Right of data portability | Right to be forgotten | Right to object to automated decision making and profiling
- Accountability and governance - PCST CoP mandates every ecosystem member to higher level of accountability and governance practice.
    - Record keeping – HAT ecosystem automatically tracks data exchange, even at a much more granular level than GDPR requires – it documents the exchange parties, time of access, detailed data points, intension and T&C, for every single transaction.
    - Data protection by design and by default - The HATDeX-serviced HAT is designed with multiple layers of protection, covering Data at Rest, Data in Transit and Data in Use. (http://www.hatdex.org/wp-content/uploads/2016/06/hatdex-briefing-Issue-2_FINAL.pdf)
- Mandatory breach notification - HAT's API driven ecosystem automatically records all exchanges breach tracking and investigation

GDPR Roundtable discussion consulted a few HAT research team members for the design of the legislation. HAT ecosystem can ensure GDPR compliance, and further mandates tighter terms than GDPR as entry requirements from all parties who wish to operate within this ecosystem following its PCST (Privacy, Confidentiality, Security and Trust) Code of Practice (http://hatcommunity.org/other-resources/).

Distributed Analytics outline

- Part I: Quick Overview of Owl
- Part II: Probabilistic Synchronous Parallel

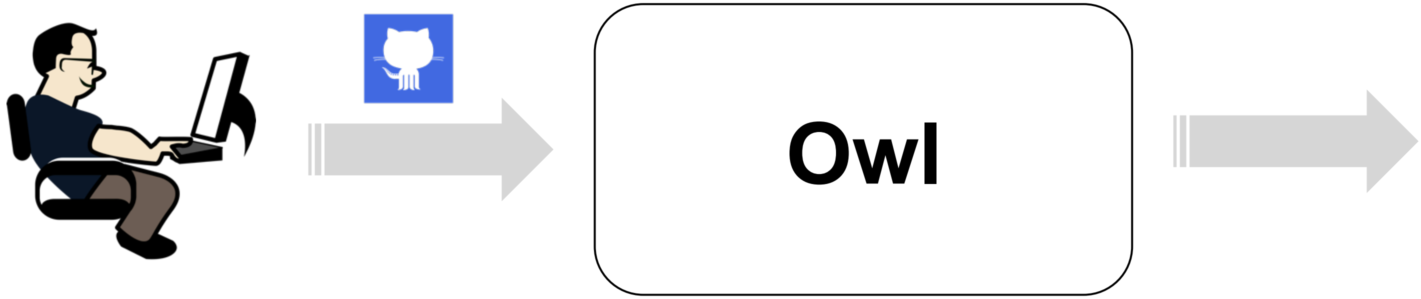
What Is Owl

- An experimental and above all scientific computing system.
- Designed in functional programming paradigm.
- Goal: as concise as Python yet as fast as C, and safe.
- A comprehensive set of classic numerical functions.
- A fundamental tooling for modern data analytics (ML & DNN).
- Native support for algorithmic differentiation, distributed & parallel computing, and GPGPU computing.

UNIVERSITY OF CAMBRIDGE

Vision Beyond Research Prototype

Write code once, then deploy it everywhere …

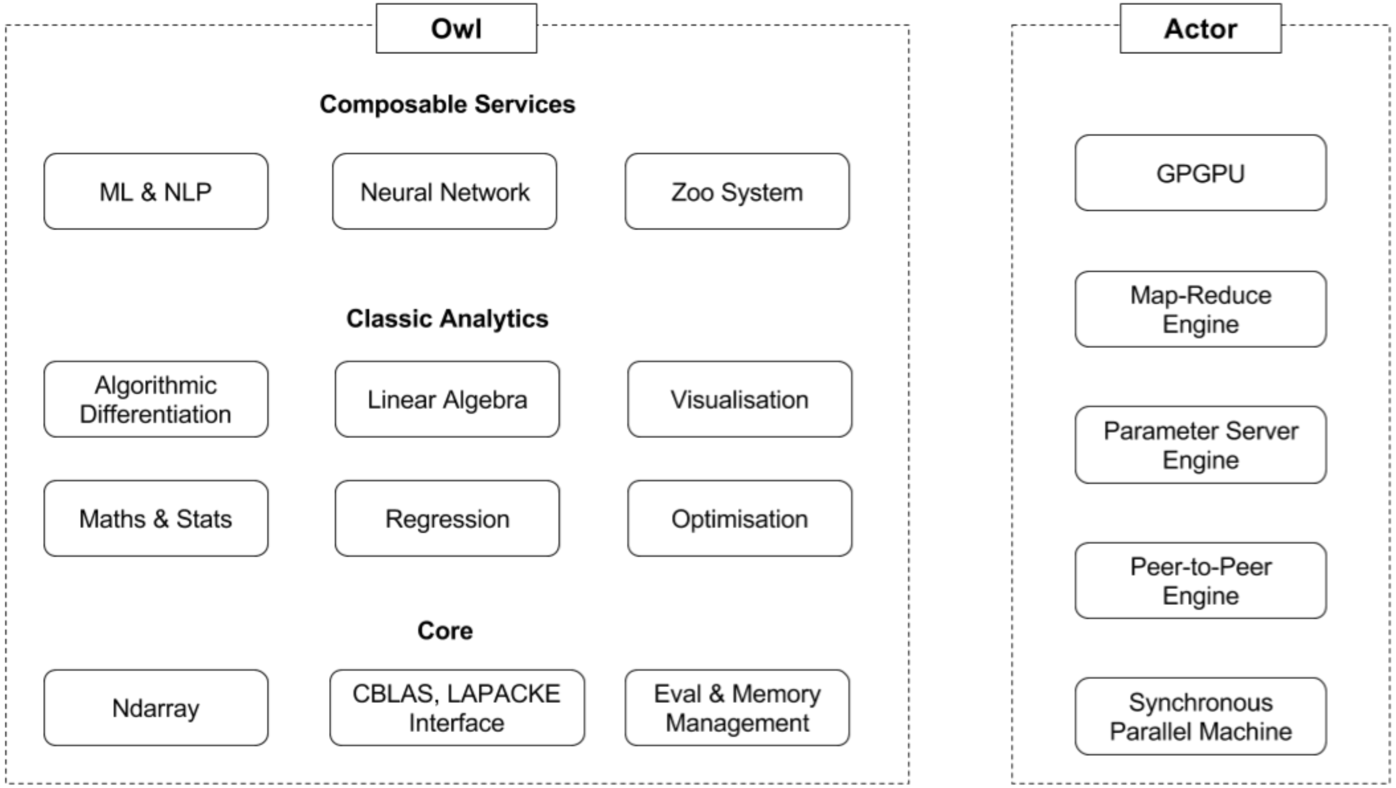


amazon web services
Xen
MirageOS Unikernel
IoT
GPU
NVIDIA CUDA
NVIDIA.COM
OpenCL

Owl system provides us a complete set of tooling from the powerful numerical supports in development to the deployment on various platforms.

UNIVERSITY OF CAMBRIDGE

Owl's Architecture



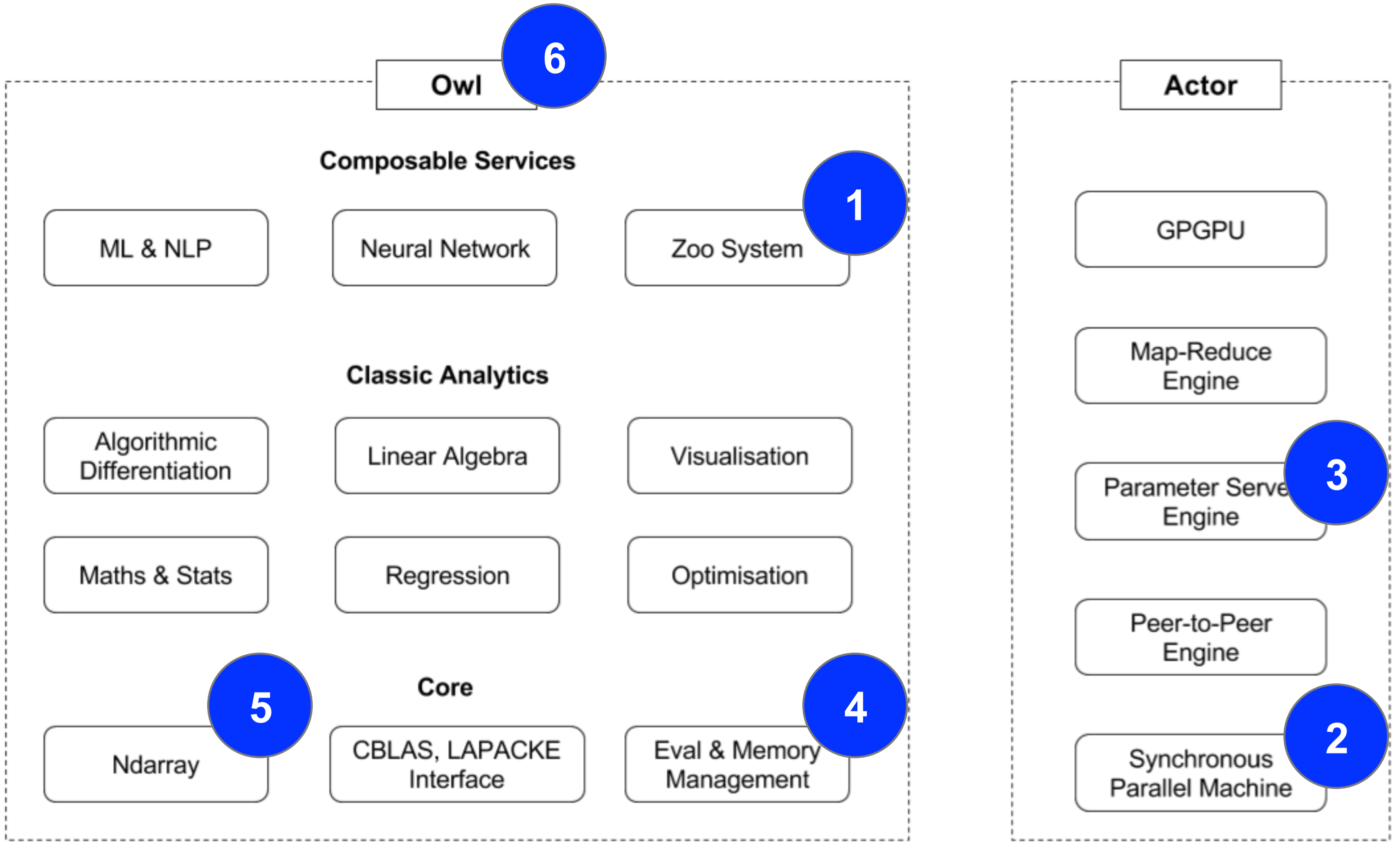
Owl + Actor = Distributed & Parallel Analytics

Owl provides numerical backend; whereas Actor implements the mechanisms of distributed and parallel computing. Two parts are connected with functors.

Various system backends allows us to write code once, then run it from cloud to edge devices, even in browsers.

Same code can run in both sequential and parallel mode with Actor engine.

Research



1. Jiaxin Zhao - *Composable Analytical Services using Session Types for Distributed Personal Data*

2. Ben Catterall - *Probabilistic Synchronous Parallel - A New Barrier Control Method for Distributed Machine Learning*

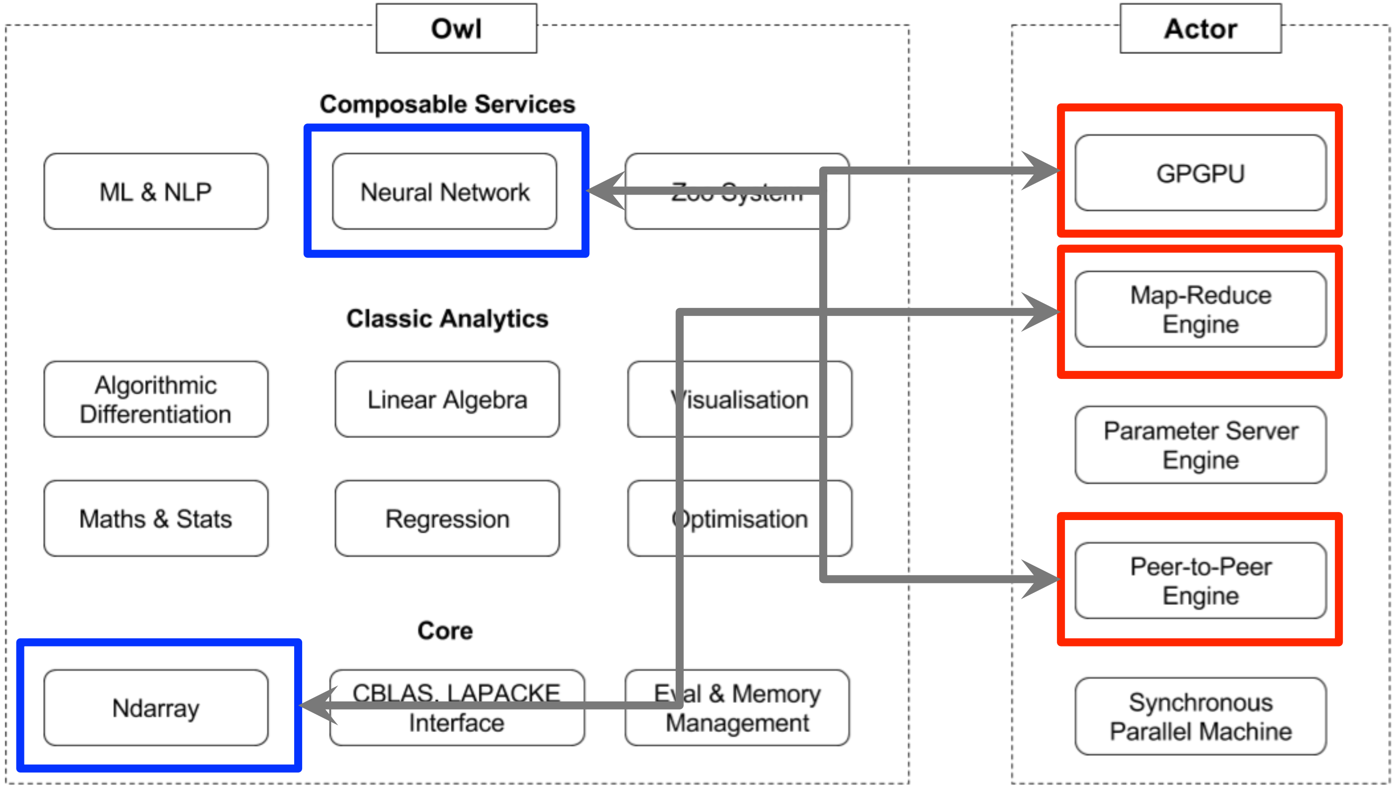
3. D.S.R Royson - *Optimising Adaptive Learning in Distributed Machine Learning*

4. Dhruv Makwana - *Memory Management using Linear Types for High-Performance GPGPU Numerical Computing*

5. Tudor Tiplea - *Deploying Browser-based Data Analytics at Network Edge*

6. Liang Wang - *Owl: A General-Purpose Numerical Library in OCaml*
Presented in ICFP’17 OCaml meeting, tutorial in CUFP’17.

Parallel and Distributed Computing



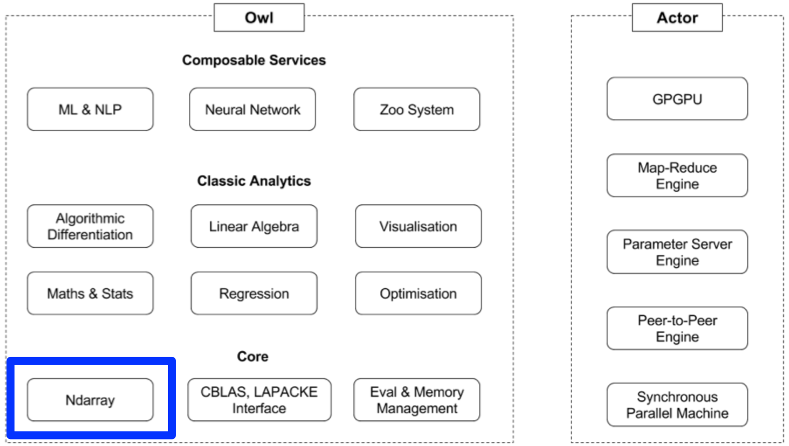
Parallel and distributed computing is achieved by composing the different data structures in Owl's core library with specific engines in Actor system.

Owl + Actor : Ndarray Example

A `map` function on local ndarray `x` in Owl looks like this

```
Dense.Ndarray.S.map sin x
```

How to implement a distributed `map` on distributed ndarray?



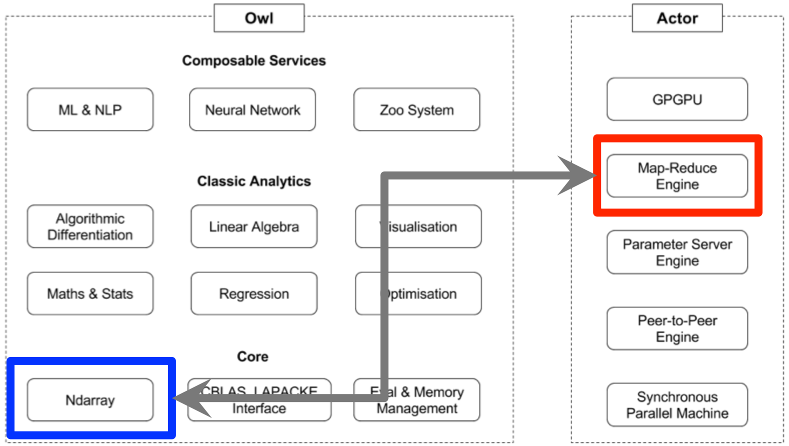
Owl + Actor : Ndarray Example

Like playing LEGO, we plug Ndarray into Distribution Engine to make a distributed Ndarray.

```
module M = Owl.Parallel.Make (Dense.Ndarray.S) (Actor.Mapre)
```

Composed by a functor in `Owl_parallel` module, which connects two systems and hides details.

```
M.map sin x
```



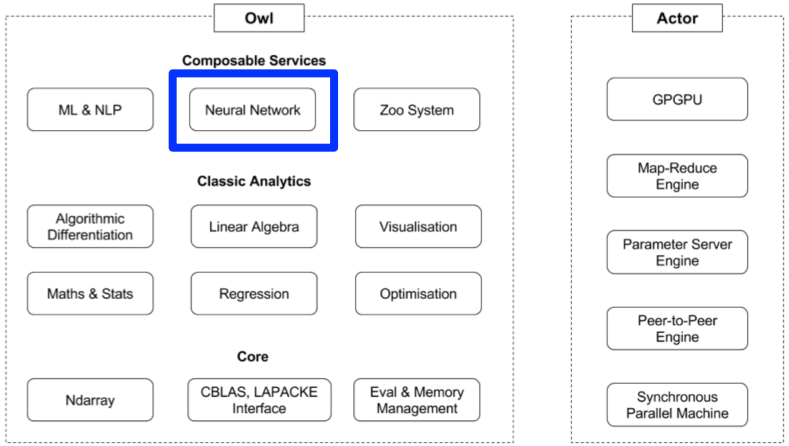
Owl + Actor : Neural Network Example

Similarly, this also applies to more advanced and complicated data structures such as neural network. This is how we define a NN in Owl:

```
let network =
  input [|28;28;1|]
  |> lambda (fun x -> Maths.(x / F 256.))
  |> conv2d [|5;5;1;32|] [|1;1|] ~act_typ:Activation.Relu
  |> max_pool2d [|2;2|] [|2;2|]
  |> dropout 0.1
  |> fully_connected 1024 ~act_typ:Activation.Relu
  |> linear 10 ~act_typ:Activation.Softmax
  |> get_network
```

Then we can perform training locally as below

```
Owl.Neural.S.Graph.train network
```



UNIVERSITY OF CAMBRIDGE

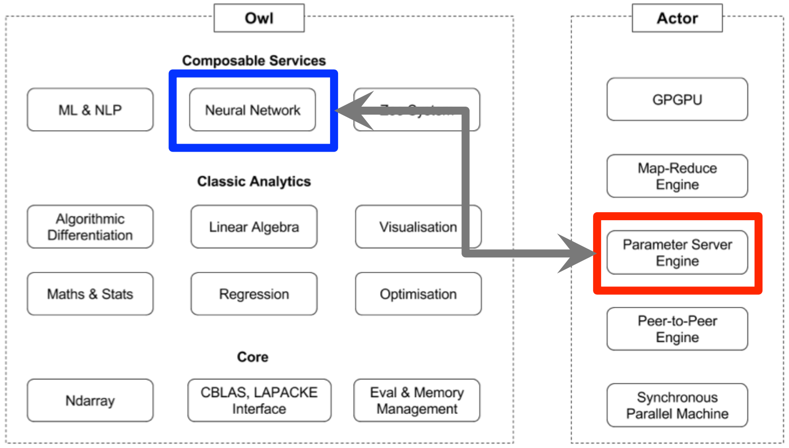
Owl + Actor : Neural Network Example

To enable the parallel training on a computer cluster, we can combine `Graph` with `Param` engine.

```
module M = Owl.Parallel.Make (Owl.Neural.S.Graph) (Actor.Param)
```

We only write code once, Owl’s functor stack generates both sequential and parallel version for us!

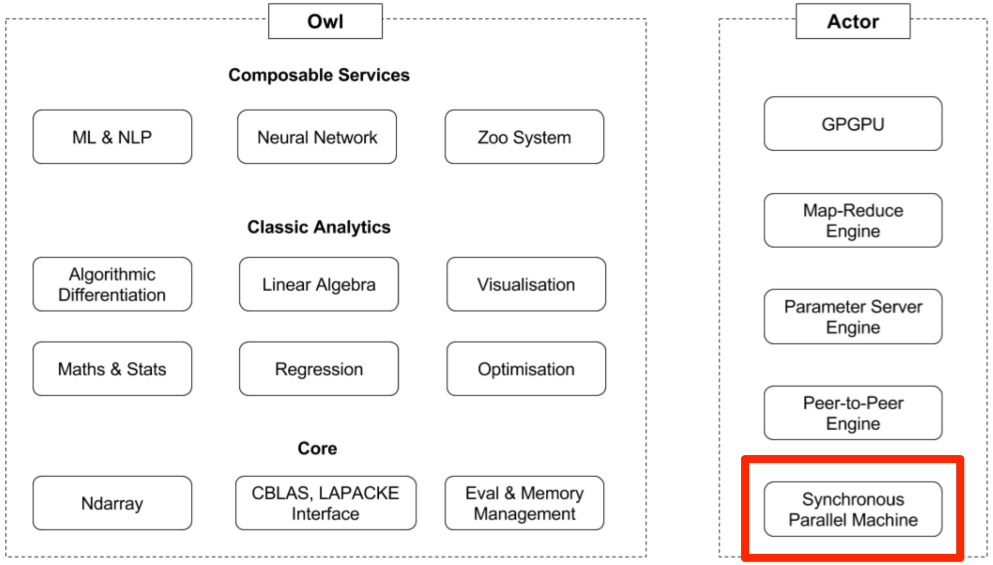
```
M.train network
```



Key to Scalability

Actor implements three engines, maybe more in future.

All reply on a module called Synchronous Parallel which handles synchronisation.



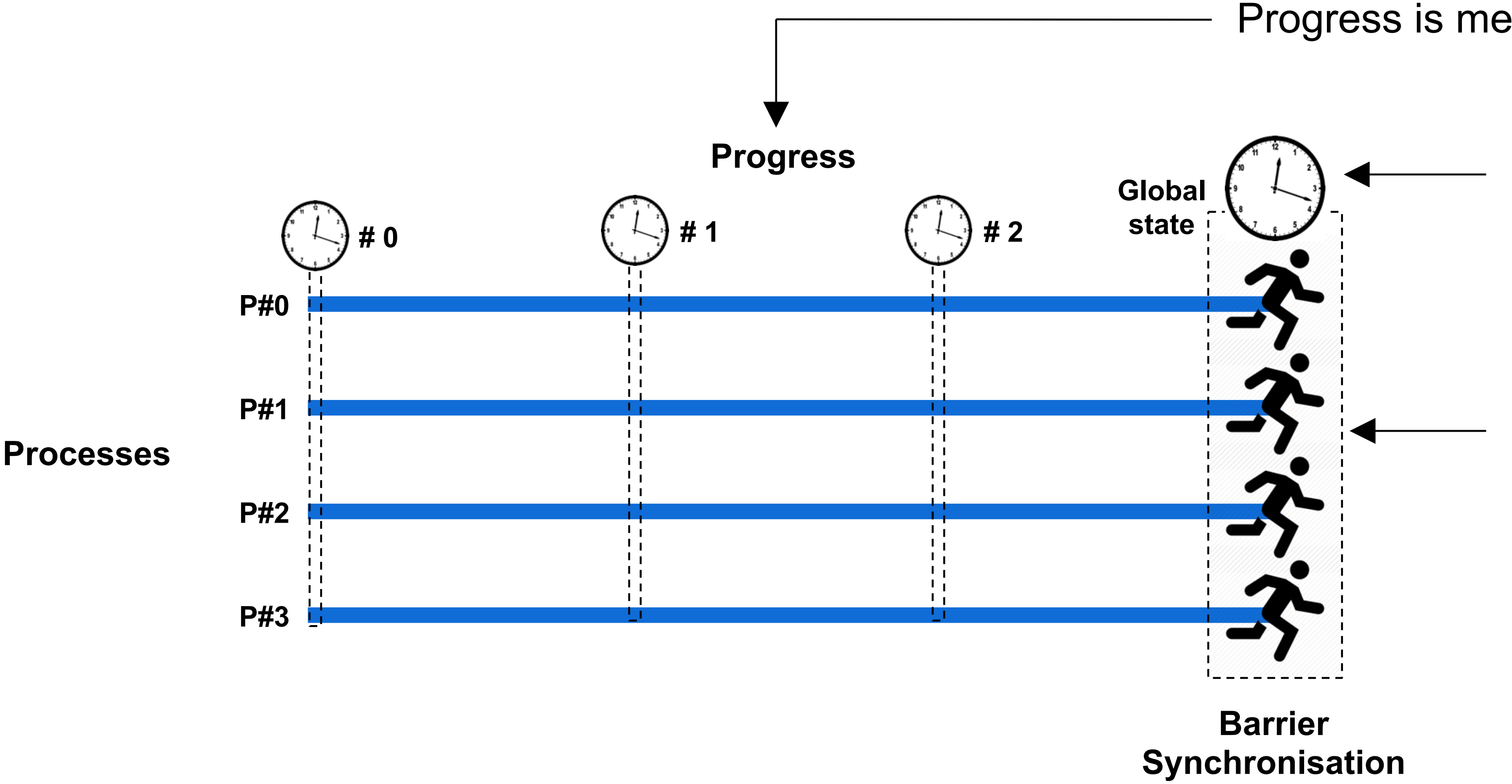
Corner stone of large scale distributed learning :)

Synchronous Parallel Machine

- An abstract computer for designing parallel algorithms.
- Three components:
    - A local processor equipped with fast memory;
    - A network that routes messages between computers;
    - A (hw/sw) component to synchronise all computers;
- Powerful model for designing and programming parallel systems, building block of Apache Hadoop, Spark, Hama, etc.

Barrier Synchronisation

The third component barrier synchronisation, is the core of SPM. It’s all about how to coordinate computation on different computers to achieve certain consistency.



Progress is measured by supersteps (iterations).

That’s where we need a central server, in order to maintain the global state.

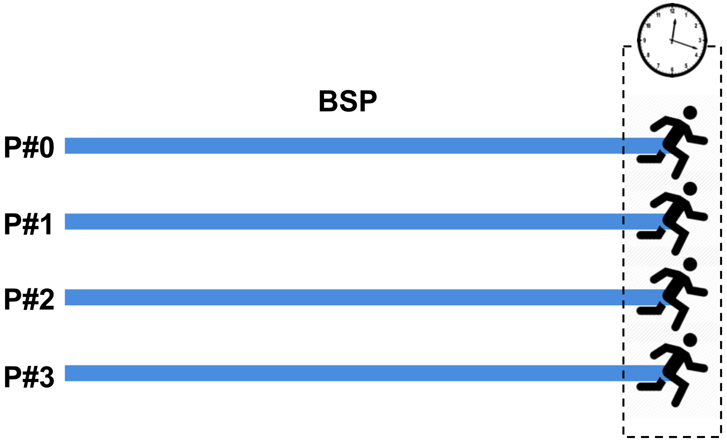
Comm may happen here, to ensure the consistency through the global state.

Consistency Is Not Free Lunch

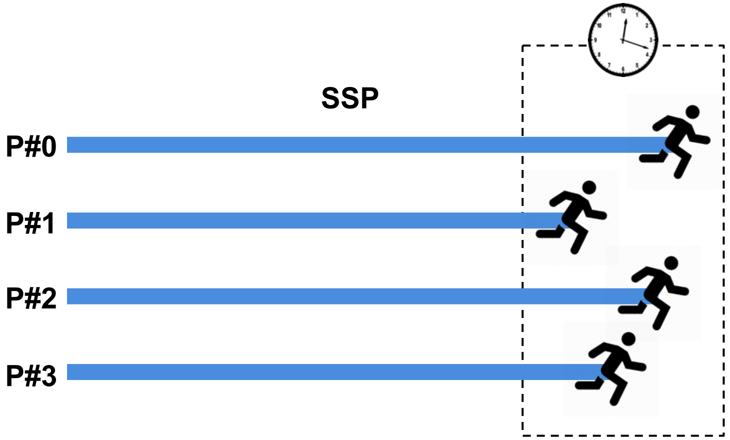
- Real world iterative learning algorithm aims fast convergence.
- Convergence rate decreases if iteration rate is slow or updates are noisy.
- Synchronisation can reduce the noise in updates (improved consistency).
- Tight synchronisation is sensitive to stragglers and has poor scalability.
- Tight synchronisation renders high communication cost in large systems.

Existing Models in Use

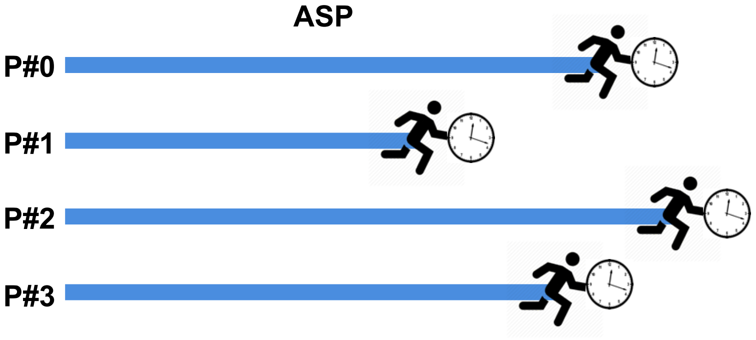
- Bulk synchronous parallel (BSP)
- Stale Synchronous parallel (SSP)
- Asynchronous parallel (ASP)



Hadoop, Spark, Parameter Server, Pregel, Owl+Actor ...

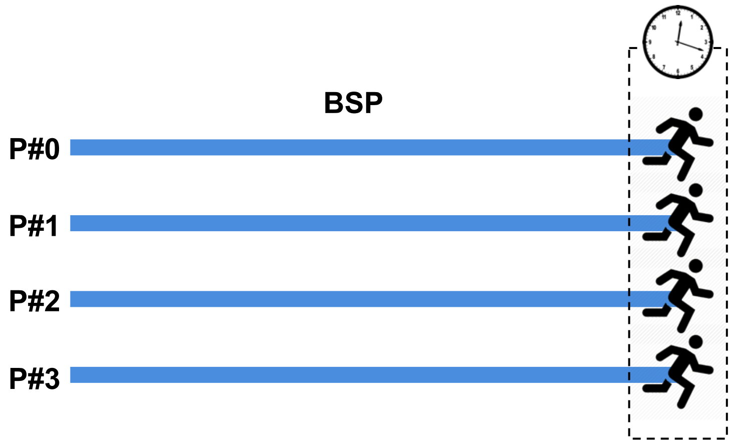


Parameter Server, Hogwild!, Cyclic Delay, Yahoo! LDA, Owl+Actor ...



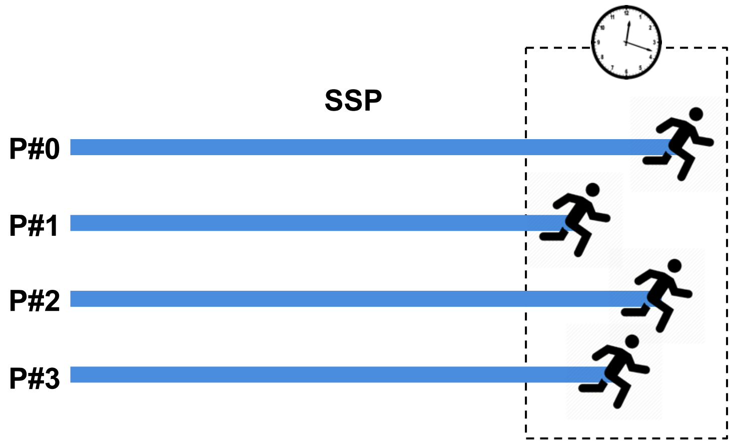
Parameter Server, Hogwild!, Yahoo! LDA, Owl+Actor ...

A 10,000 Foot View



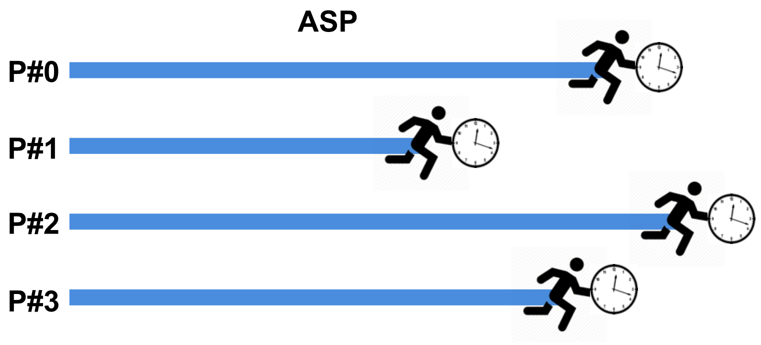
Most strict lockstep synchronisation; all the nodes are coordinated by a central server.

BSP is sensitive to stragglers so is very slow. But it is simple due to its deterministic nature, easy to write application on top of it.



SSP relaxes consistency by allowing difference in iteration rate. The difference is controlled by the bounded staleness.

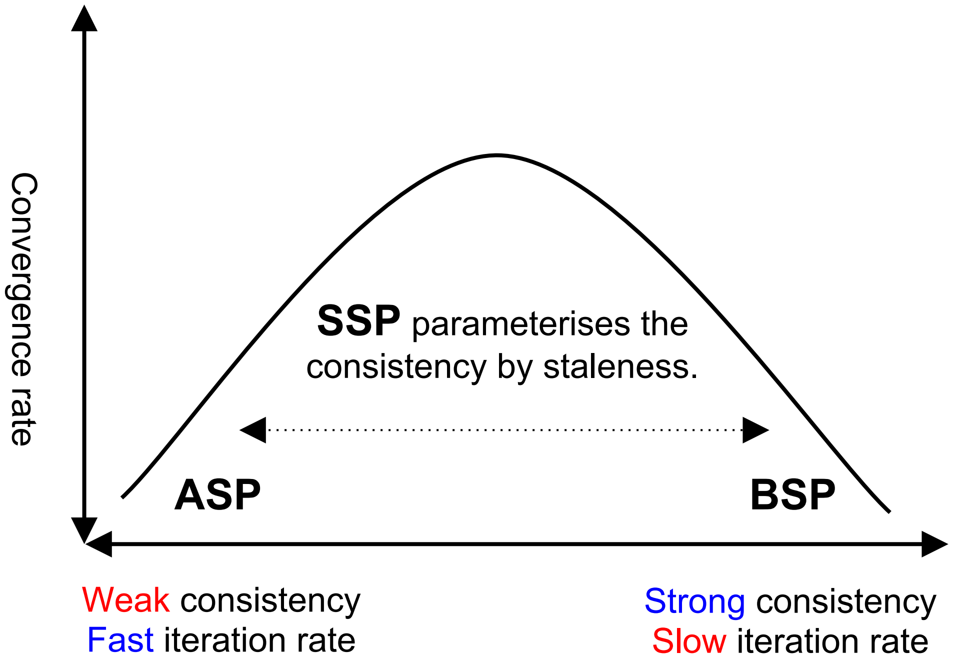
SSP is supposed to mitigate the negative effects of stragglers. But the server still requires global state.



Least strict synchronisation, no communication among workers for barrier synchronisation all all.

Every computer can progress as fast as it can. It is fast and scalable, but often produces noisy updates. No theoretical guarantees on consistency and algorithm’s convergence.

Consistency vs. Iteration Rate



We must balance consistency vs. iteration rate , we can think in the following way to understand the trade-off:

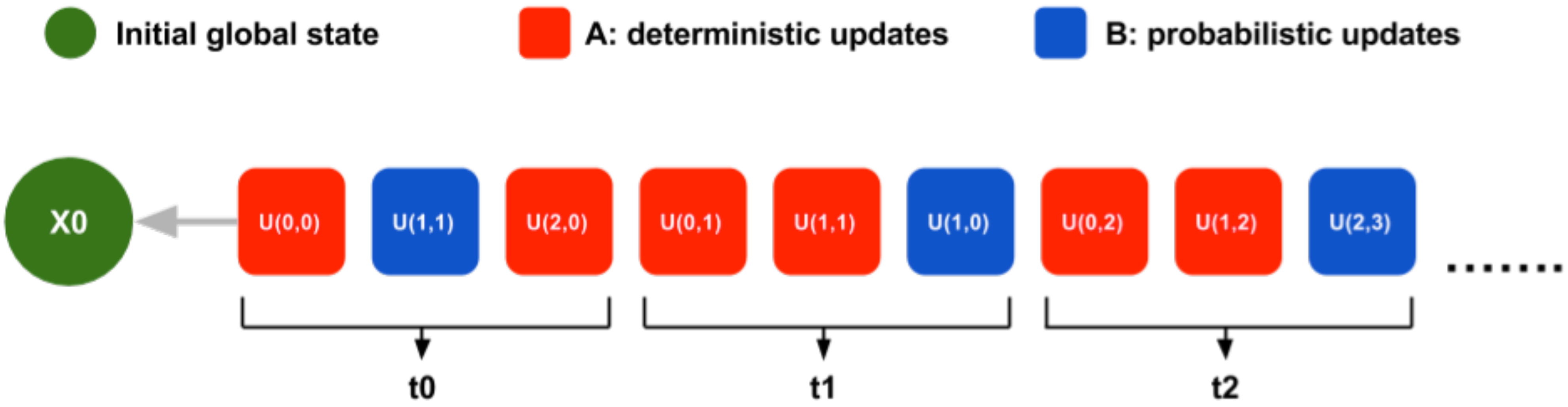
Convergence ∝ Consistency Degree x Iteration Rate

SSP aims to cover this spectrum between BSP and ASP by parameterising the staleness.

Question: Is SSP really a generalisation of of BSP and ASP?
BSP and SSP are both centralised whereas ASP are fully distributed. It feels like SSP has missed something in this spectrum. What is it?

UNIVERSITY OF CAMBRIDGE

Analytical Model



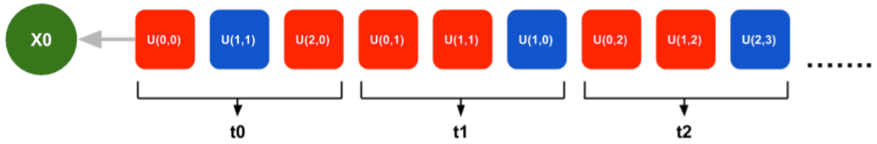
$$\mathbf{x}_0 \mathrel{+}= \left[\sum_{\mathcal{A}} \mathbf{u}_{\mathbf{p},\mathbf{t}}\right] + \left[\sum_{\mathcal{B}} \mathbf{u}_{\mathbf{p},\mathbf{t}}\right]$$

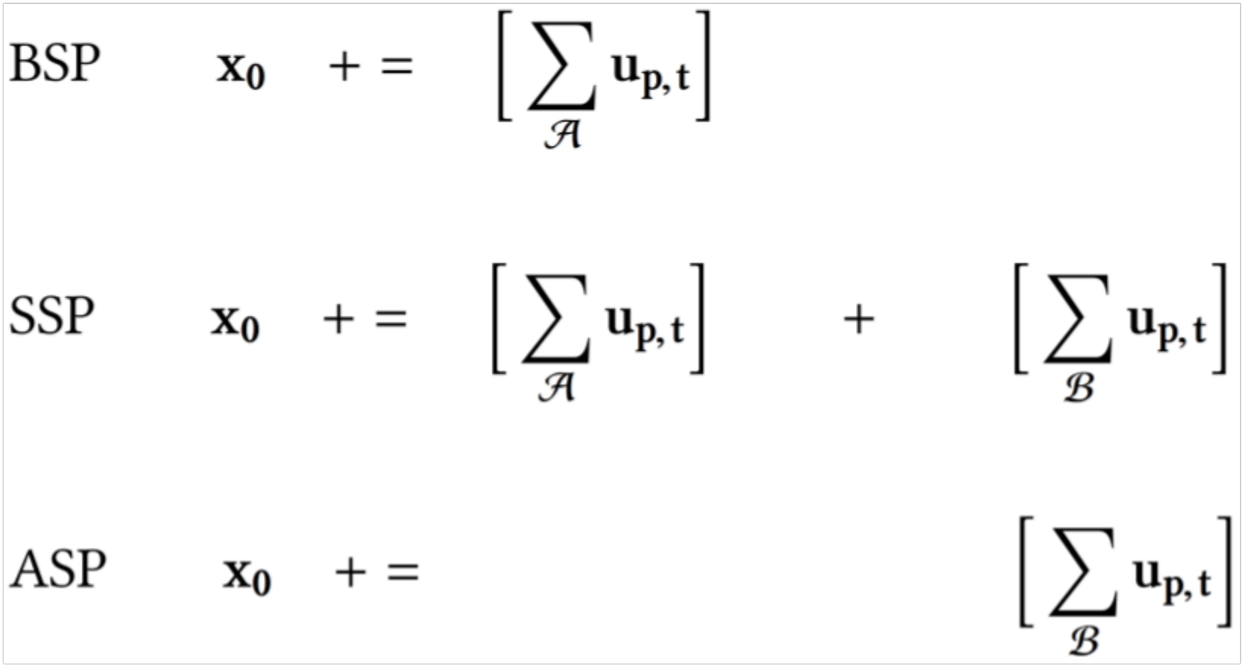
`u(p,t)` is `update(node id, timestamp)`, sum over all the nodes and clock ticks ...

The model is simple: a sequence of updates applying to an initial global state **x0**. The updates can be “noisy” hence are divided into two sets.

Although simple, it can model most iterative learning algorithms.

Decompose Synchronous Parallel Machine





BSP	$\mathbf{x}_0$	$+=$	$\left[\sum_{\mathcal{A}} \mathbf{u}_{p,t}\right]$		
SSP	$\mathbf{x}_0$	$+=$	$\left[\sum_{\mathcal{A}} \mathbf{u}_{p,t}\right]$	$+$	$\left[\sum_{\mathcal{B}} \mathbf{u}_{p,t}\right]$
ASP	$\mathbf{x}_0$	$+=$			$\left[\sum_{\mathcal{B}} \mathbf{u}_{p,t}\right]$

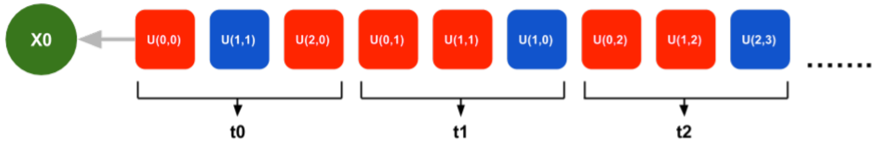
Then we use the analytical model to express each synchronous parallel machine on the left.

The formulation reveals some very interesting structures from a system design perspective.

Let’s decompose these machines.

Decompose Synchronous Parallel Machine

	Consistency	Synchronisation		
BSP	$\mathbf{x_0} \ +=$	$\left[\sum_{\mathcal{A}} \mathbf{u_{p,t}}\right]$		
SSP	$\mathbf{x_0} \ +=$	$\left[\sum_{\mathcal{A}} \mathbf{u_{p,t}}\right]$	$+$	$\left[\sum_{\mathcal{B}} \mathbf{u_{p,t}}\right]$
ASP	$\mathbf{x_0} \ +=$			$\left[\sum_{\mathcal{B}} \mathbf{u_{p,t}}\right]$

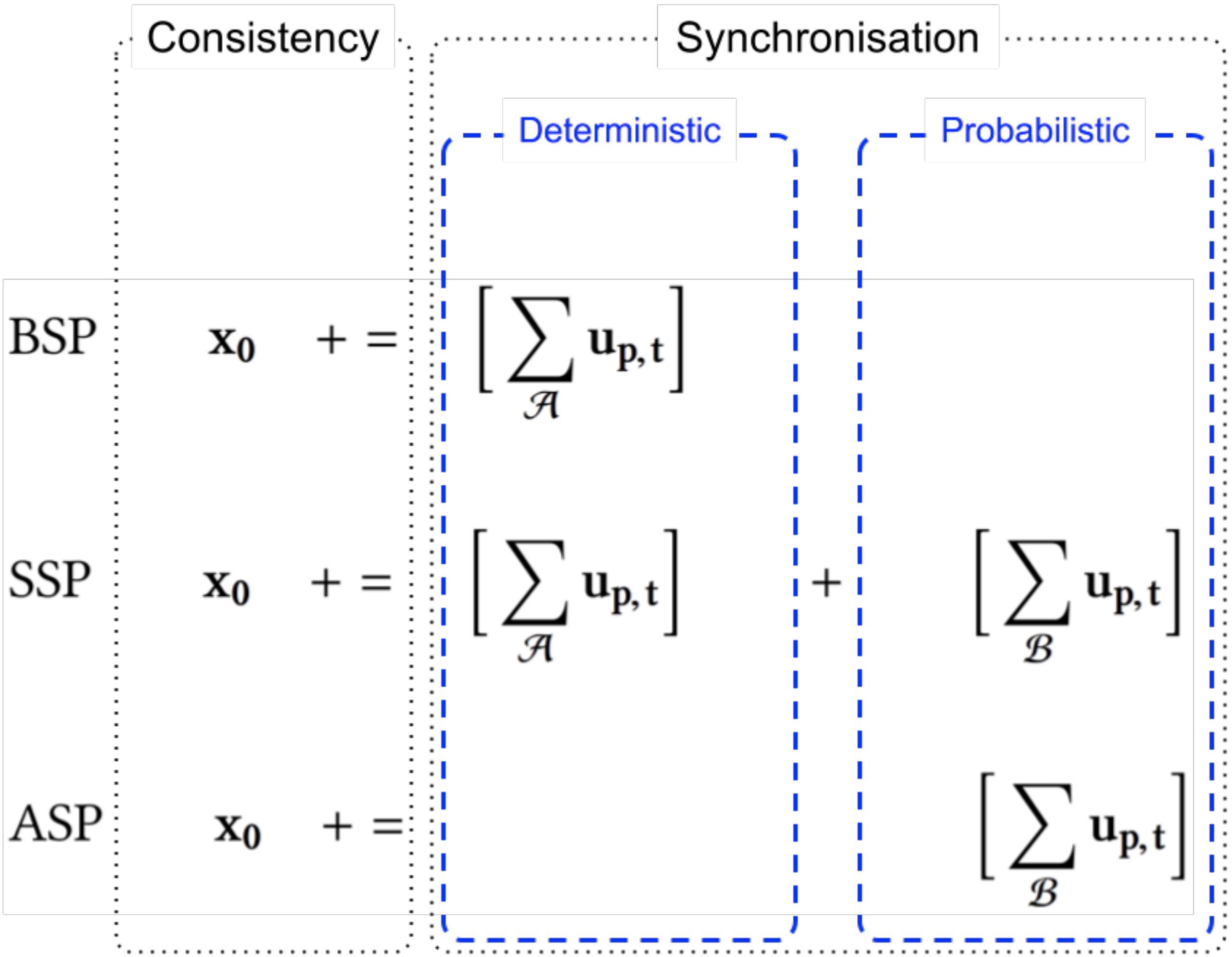


Left part deals with the Consistency. **+=** operator is the server logic about how to incorporate updates submitted to the central server into the global state.

Right part deals with synchronisation, computers either communicate to each other or contact the central server to coordinate their progress.

UNIVERSITY OF CAMBRIDGE

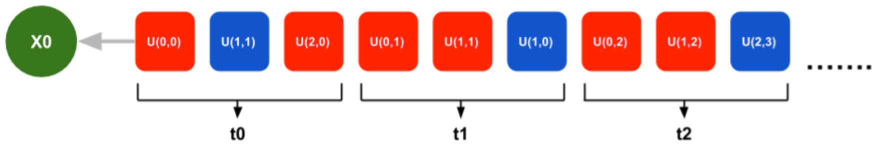
Decompose Synchronous Parallel Machine



BSP $$\mathbf{x_0} \mathrel{+}= \left[\sum_{\mathcal{A}} \mathbf{u_{p,t}}\right]$$

SSP $$\mathbf{x_0} \mathrel{+}= \left[\sum_{\mathcal{A}} \mathbf{u_{p,t}}\right] + \left[\sum_{\mathcal{B}} \mathbf{u_{p,t}}\right]$$

ASP $$\mathbf{x_0} \mathrel{+}= \left[\sum_{\mathcal{B}} \mathbf{u_{p,t}}\right]$$



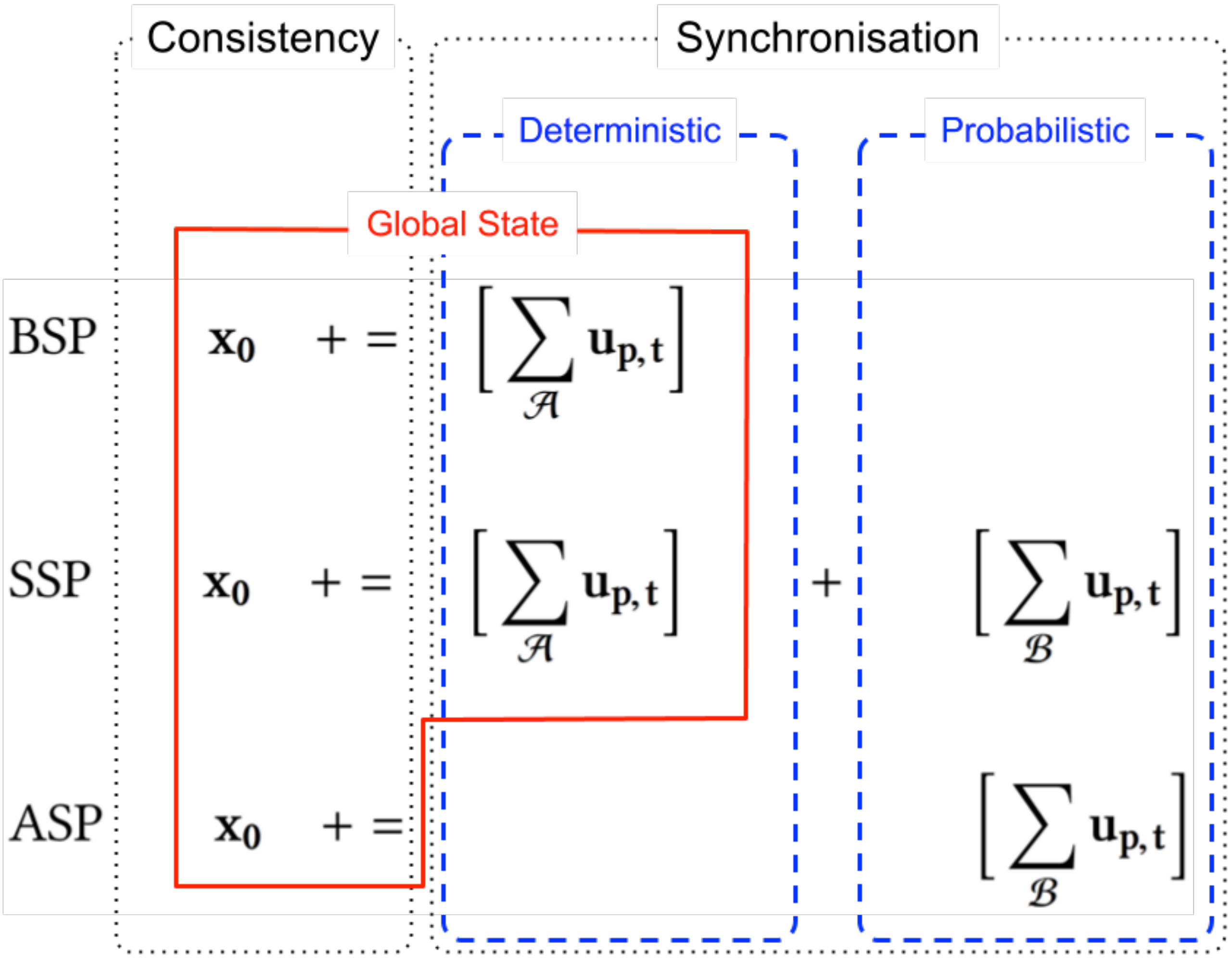
Inside the synchronisation part, the synchronous parallel machine processes two types of updates.

The deterministic one is those we always expect if everything goes well as in BSP.

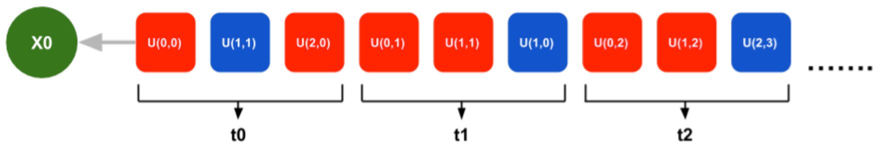
The probabilistic one is those out of order updates due to packet loss, network delay, node failure, and etc.

UNIVERSITY OF CAMBRIDGE

Decompose Synchronous Parallel Machine



Consistency
Synchronisation
Deterministic
Probabilistic
Global State
BSP
$\mathbf{x}_0 \quad += \quad \left[\sum_{\mathcal{A}} \mathbf{u}_{p,t}\right]$
SSP
$\mathbf{x}_0 \quad += \quad \left[\sum_{\mathcal{A}} \mathbf{u}_{p,t}\right] + \left[\sum_{\mathcal{B}} \mathbf{u}_{p,t}\right]$
ASP
$\mathbf{x}_0 \quad += \quad \left[\sum_{\mathcal{B}} \mathbf{u}_{p,t}\right]$



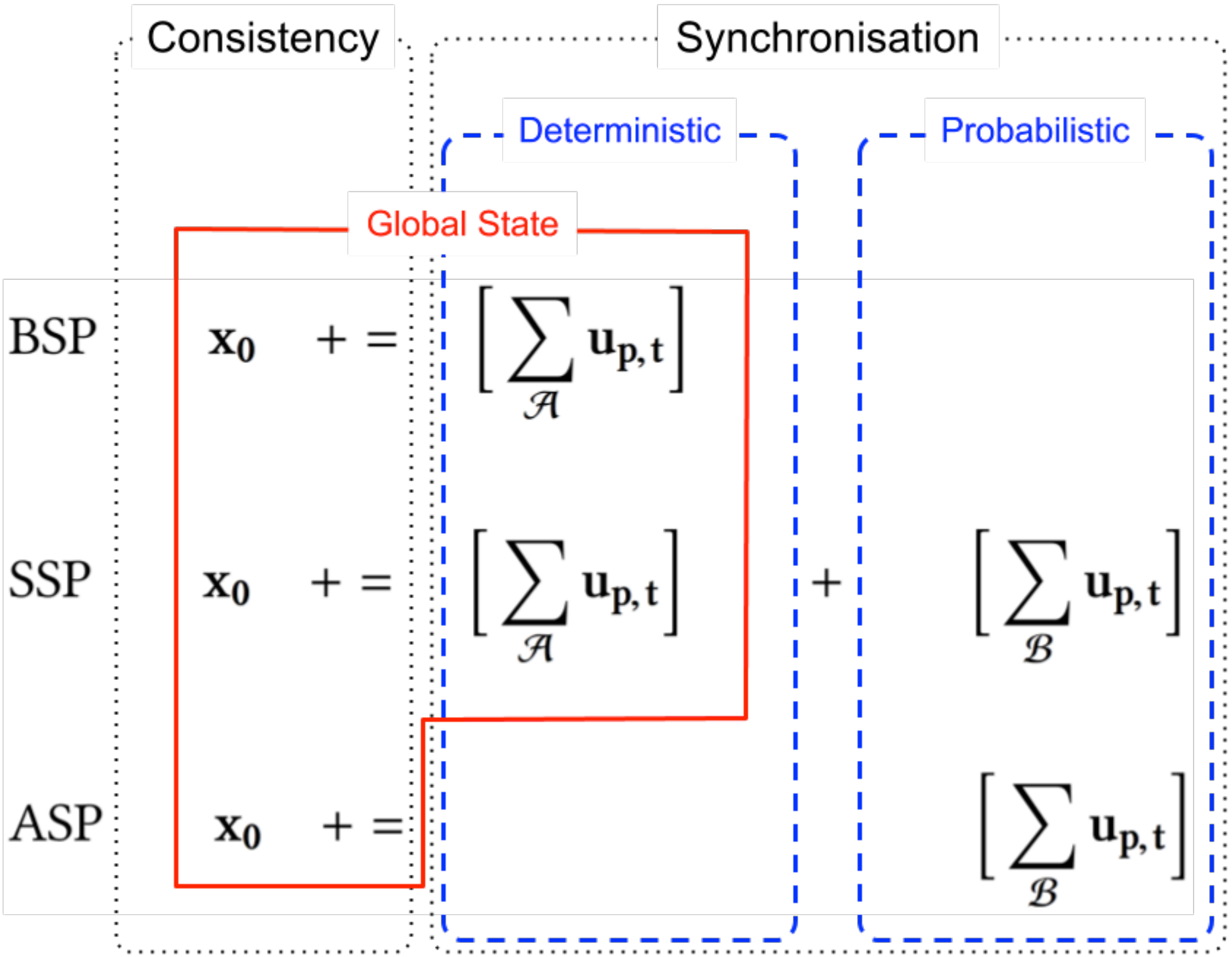
The global state is maintained by a logic central server. In a distributed system, this is often the bottleneck and single point of failure.

Moreover, note that the server couples the consistency with the synchronisation two parts, for BSP and SSP.

ASP avoids such coupling by giving up synchronisation, consistency completely.

UNIVERSITY OF CAMBRIDGE

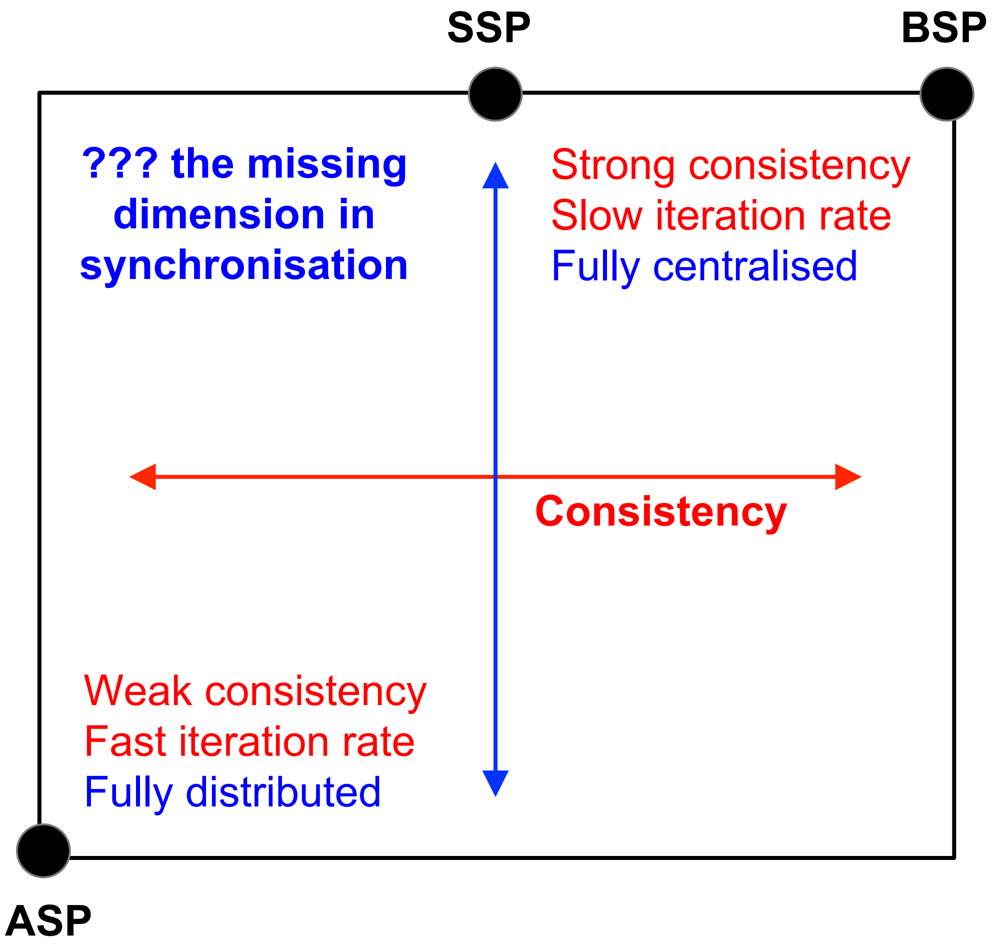
Missing Dimension in Design Space



$$x_0 \mathrel{+}= \left[\sum_{\mathcal{A}} u_{p,t}\right]$$

$$x_0 \mathrel{+}= \left[\sum_{\mathcal{A}} u_{p,t}\right] + \left[\sum_{\mathcal{B}} u_{p,t}\right]$$

$$x_0 \mathrel{+}= \left[\sum_{\mathcal{B}} u_{p,t}\right]$$



??? the missing dimension in synchronisation

Strong consistency
Slow iteration rate
Fully centralised

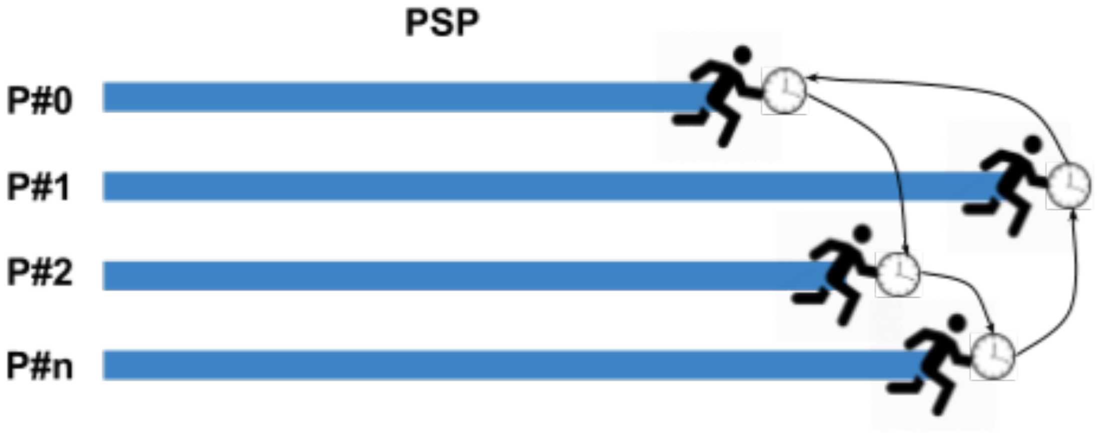
Weak consistency
Fast iteration rate
Fully distributed

Key Insights from Decomposition

- Is it necessary to couple the consistency with synchronisation? No
- Is it necessary to give up synchronisation completely in order to decouple consistency and synchronisation? No
- Is it necessary to divide the updates into deterministic and probabilistic two parts? No
- Is SSP really a generalisation of BSP and ASP, then what is the missing dimension in the design space? Completeness

Probabilistic Synchronous Parallel

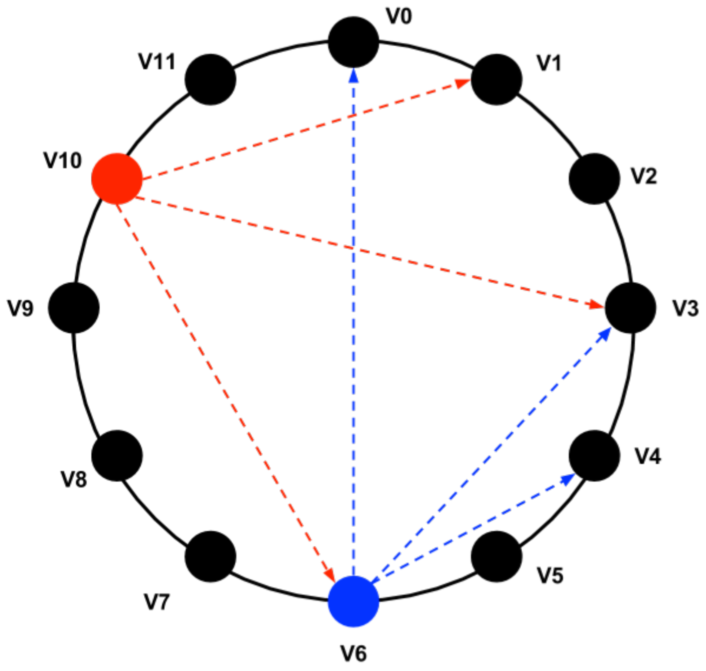
- Core idea: combine both deterministic and probabilistic components, replace it with a sample distribution.
- Each computer synchronises with a small group of others and the consistency is only enforced within the group.
- The server decides how to incorporate the submitted updates.



No central server to coordinate them, instead each node synchronises within their groups. The consistency “propagates” by the possible overlapping of different groups.

Sampling Primitive

- How to implement PSP atop of current data analytics frameworks?
    - Quite straightforward, add a new primitive - `sample`
- How to guarantee the random sampling?
    - Organise the nodes into a structural overlay, e.g. DHT



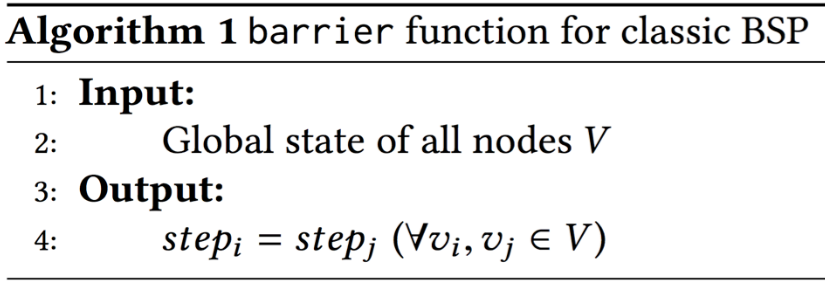
The random sampling is based on the fact that node identifiers are uniformly distributed in a name space.

Node can estimate the population size based on the allocated ID density in the name space.

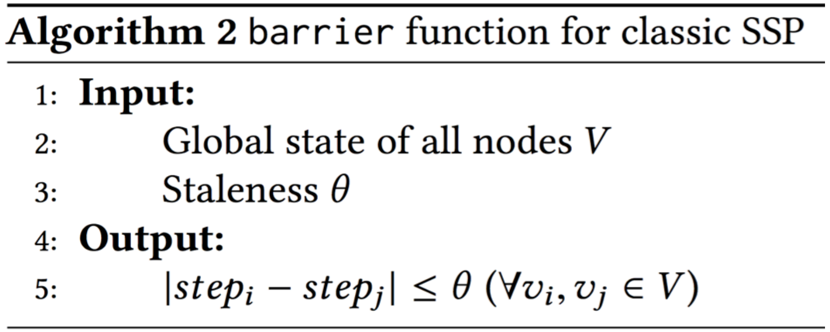
UNIVERSITY OF CAMBRIDGE

As A Higher-Order Function

- PSP is a generalisation of BSP, SSP, ASP.
- PSP can be applied to other synchronous machine as a higher-order function, to further derive a fully distributed version.

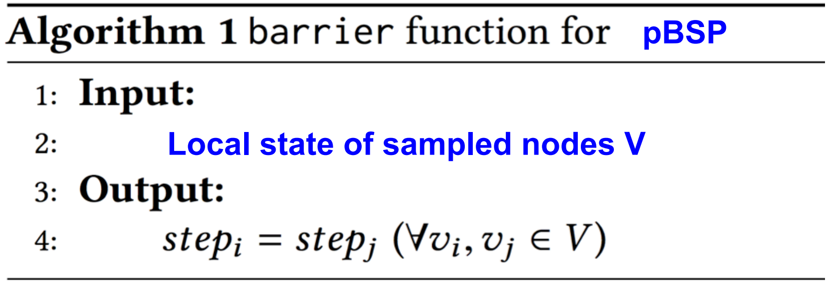


```
**Algorithm 1** barrier function for classic BSP
Input:
    Global state of all nodes $V$
Output:
    $step_i = step_j \ (\forall v_i, v_j \in V)$
```

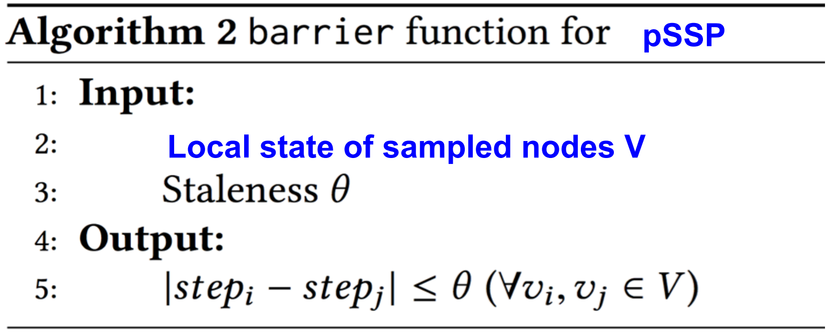


```
**Algorithm 2** barrier function for classic SSP
Input:
    Global state of all nodes $V$
    Staleness $\theta$
Output:
    $|step_i - step_j| \leq \theta \ (\forall v_i, v_j \in V)$
```

`Sample function ○ Barrier function`



```
**Algorithm 1** barrier function for pBSP
Input:
    Local state of sampled nodes V
Output:
    $step_i = step_j \ (\forall v_i, v_j \in V)$
```



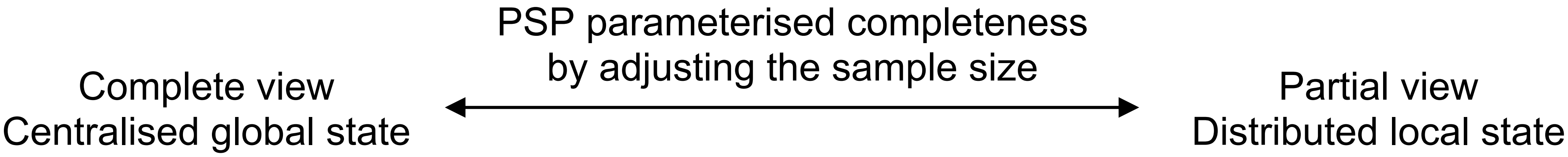
```
**Algorithm 2** barrier function for pSSP
Input:
    Local state of sampled nodes V
    Staleness $\theta$
Output:
    $|step_i - step_j| \leq \theta \ (\forall v_i, v_j \in V)$
```

Fully compatible with existing systems due to its generality.

UNIVERSITY OF CAMBRIDGE

A Newly Discovered Dimension

- The degree of consistency is enforced upon a sample rather than the whole population.
- PSP adds a new dimension to the existing synchronous parallel models, i.e. the degree of completeness in a sample, which renders a distribution of degree of consistency.
- PSP allows us to parameterise the degree of completeness (or level of coordination), ranging from a fully centralised system to a fully distributed one.

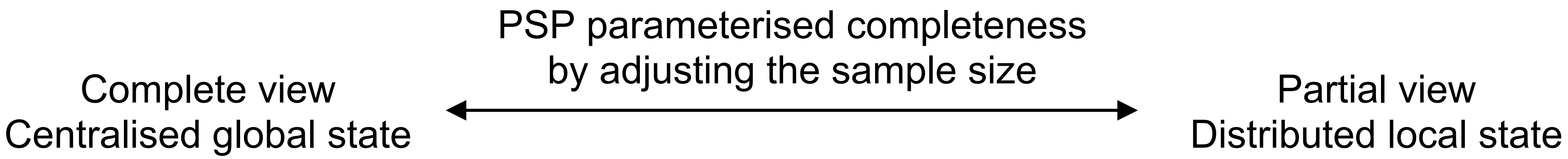


Complete view
Centralised global state

PSP parameterised completeness
by adjusting the sample size

Partial view
Distributed local state

Factorise Consistency by Completeness



PSP parameterised completeness by adjusting the sample size

Complete view
Centralised global state

Partial view
Distributed local state

Because of this new dimension, Consistency Degree is now factorised into two parts.

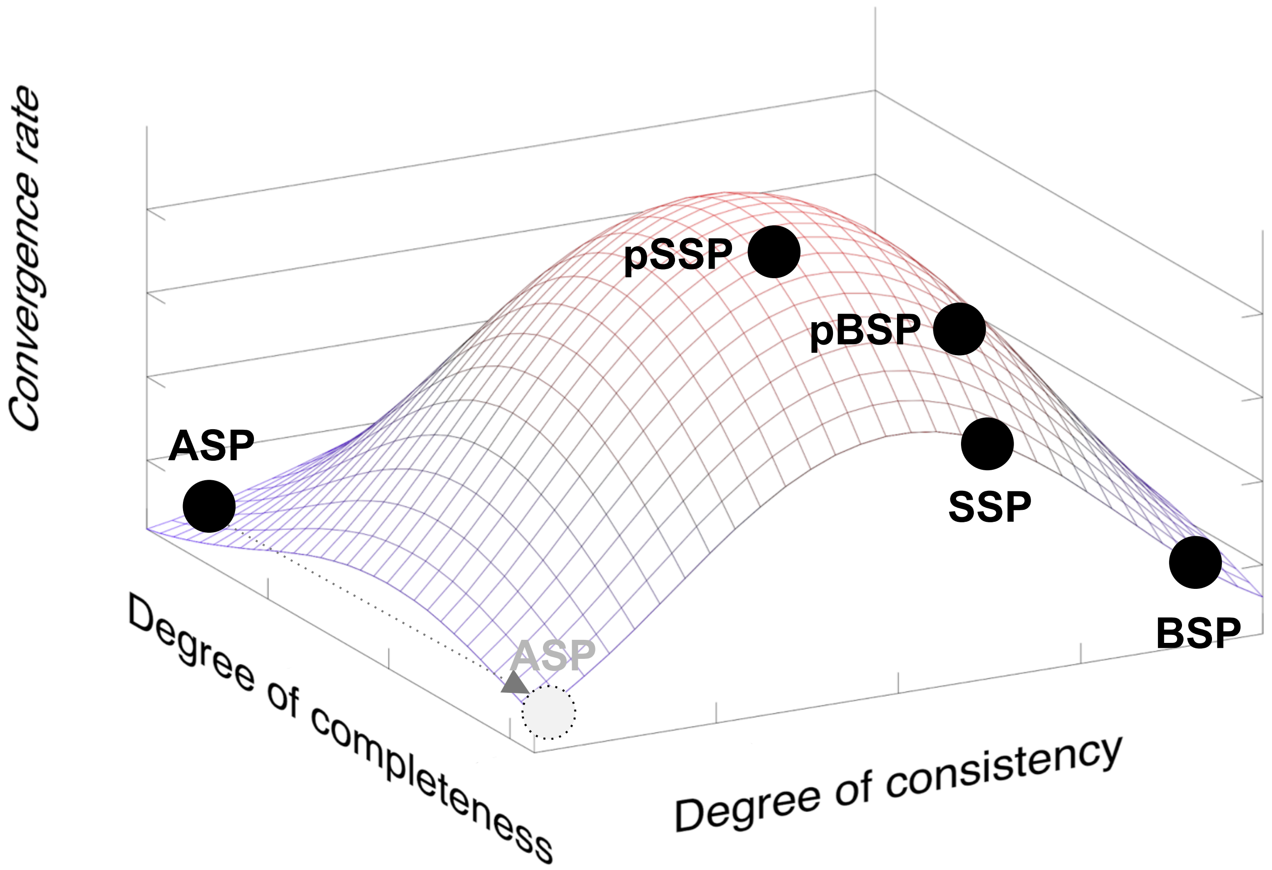
Convergence ∝ Consistency Degree x Iteration Rate

↓

Convergence ∝ Consistency Degree in Sample x Completeness of Sample x Iteration Rate

Trade-off in a Larger Design Space

Convergence = Consistency Degree in Sample x Completeness of Sample x Iteration Rate



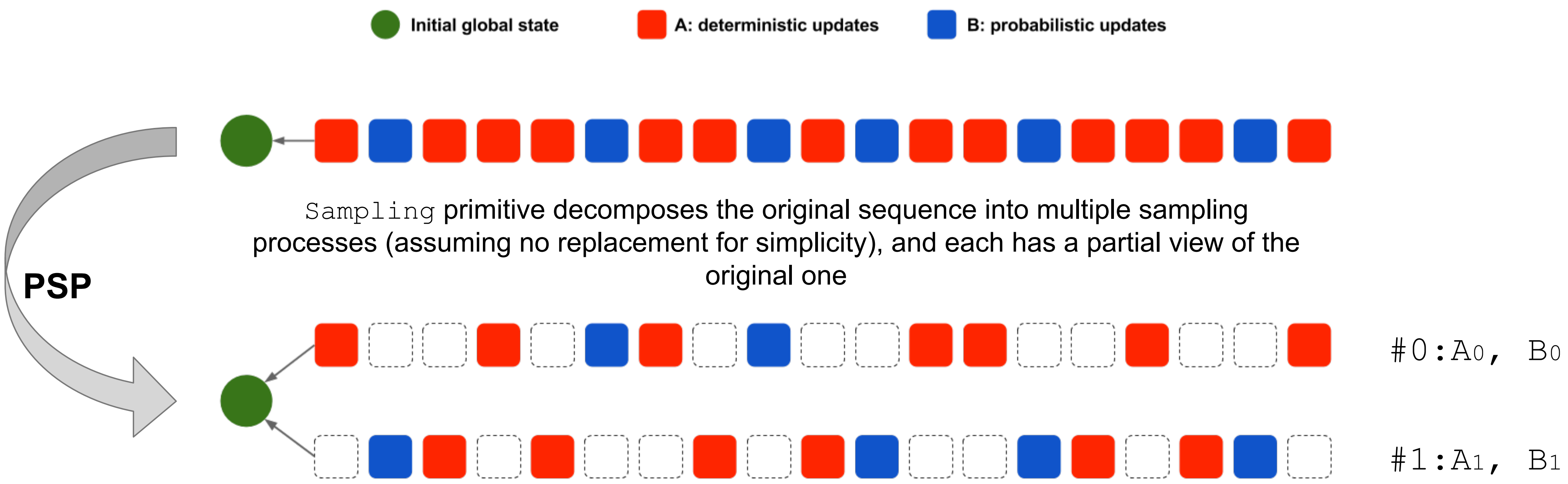
The new dimension allows us to explore a larger design space, which further makes it possible to find better a trade-off to achieve better convergence rate.

ASP doesn’t really fall on the same *Convergence - Consistency* plane.

Degree of consistency along the new dimension becomes a distribution now.

Note this figure is plotted with Owl.

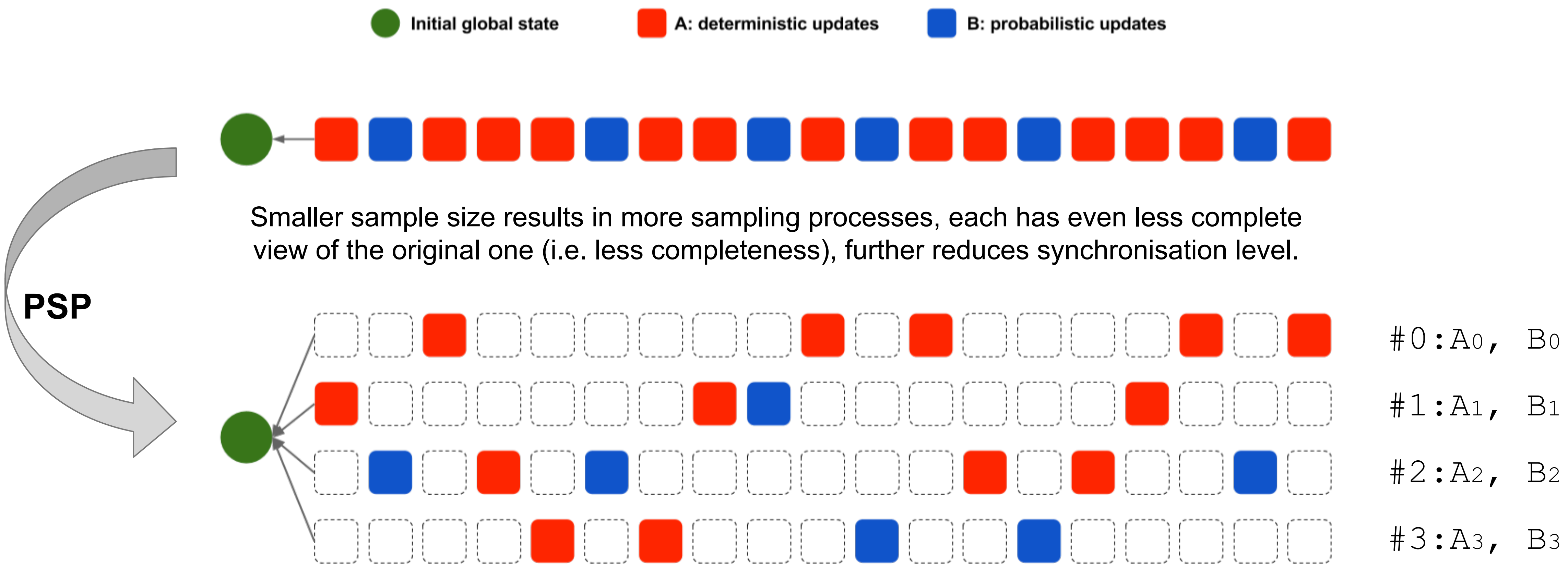
Effects of Sampling Primitive



`Sampling` primitive decomposes the original sequence into multiple sampling processes (assuming no replacement for simplicity), and each has a partial view of the original one

$$\text{PSP} \qquad \mathbf{x_0} \quad += \quad \sum_{\forall i \in N} \left[\sum_{\mathcal{A}_i} \mathbf{u_{p,t}} \quad + \quad \sum_{\mathcal{B}_i} \mathbf{u_{p,t}} \right]$$

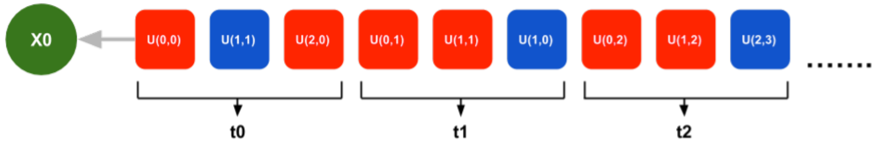
Effects of Smaller Samples



Smaller sample size results in more sampling processes, each has even less complete view of the original one (i.e. less completeness), further reduces synchronisation level.

$$\text{PSP} \qquad \mathbf{x_0} \mathrel{+}= \sum_{\forall i \in N} \left[\sum_{\mathcal{A}_i} \mathbf{u_{p,t}} + \sum_{\mathcal{B}_i} \mathbf{u_{p,t}} \right]$$

Revisit System Decomposition

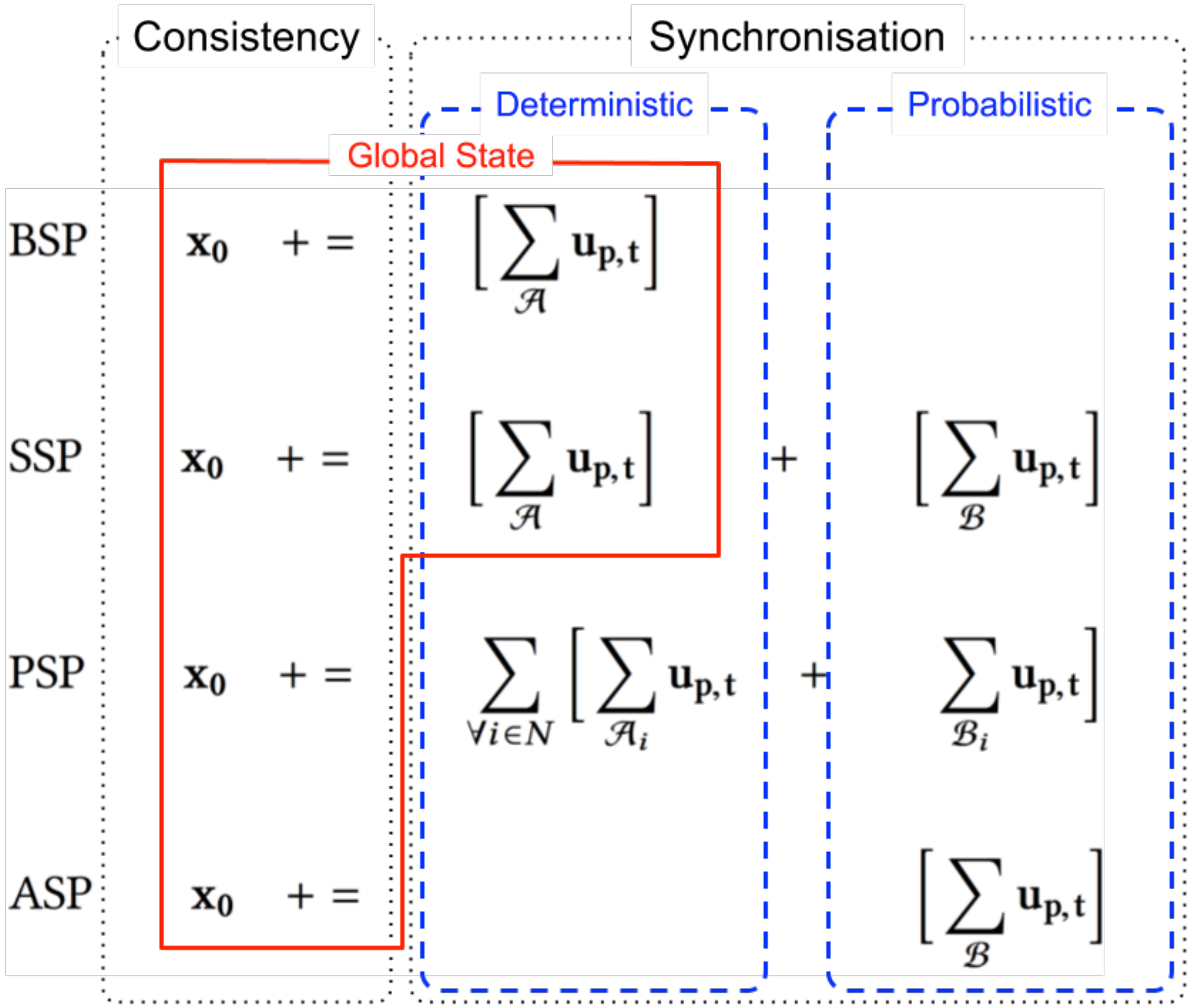


$$
\begin{array}{llll}
\text{BSP} & \mathbf{x_0} & += & \left[\sum_{\mathcal{A}} \mathbf{u_{p,t}}\right] \\
\text{SSP} & \mathbf{x_0} & += & \left[\sum_{\mathcal{A}} \mathbf{u_{p,t}}\right] + \left[\sum_{\mathcal{B}} \mathbf{u_{p,t}}\right] \\
\text{PSP} & \mathbf{x_0} & += & \sum_{\forall i \in N} \left[\sum_{\mathcal{A}_i} \mathbf{u_{p,t}} + \sum_{\mathcal{B}_i} \mathbf{u_{p,t}}\right] \\
\text{ASP} & \mathbf{x_0} & += & \left[\sum_{\mathcal{B}} \mathbf{u_{p,t}}\right]
\end{array}
$$

PSP now bridges the gap between SSP and ASP by unifying the deterministic and probabilistic components with the updates of a distribution of clock ticks.

UNIVERSITY OF CAMBRIDGE

Revisit System Decomposition

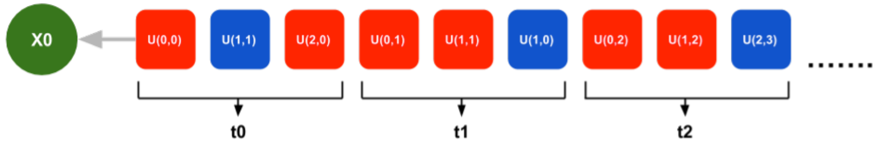


BSP $\mathbf{x_0} \mathrel{+}= \left[\sum_{\mathcal{A}} \mathbf{u_{p,t}}\right]$

SSP $\mathbf{x_0} \mathrel{+}= \left[\sum_{\mathcal{A}} \mathbf{u_{p,t}}\right] + \left[\sum_{\mathcal{B}} \mathbf{u_{p,t}}\right]$

PSP $\mathbf{x_0} \mathrel{+}= \sum_{\forall i \in N} \left[\sum_{\mathcal{A}_i} \mathbf{u_{p,t}} + \sum_{\mathcal{B}_i} \mathbf{u_{p,t}}\right]$

ASP $\mathbf{x_0} \mathrel{+}= \left[\sum_{\mathcal{B}} \mathbf{u_{p,t}}\right]$



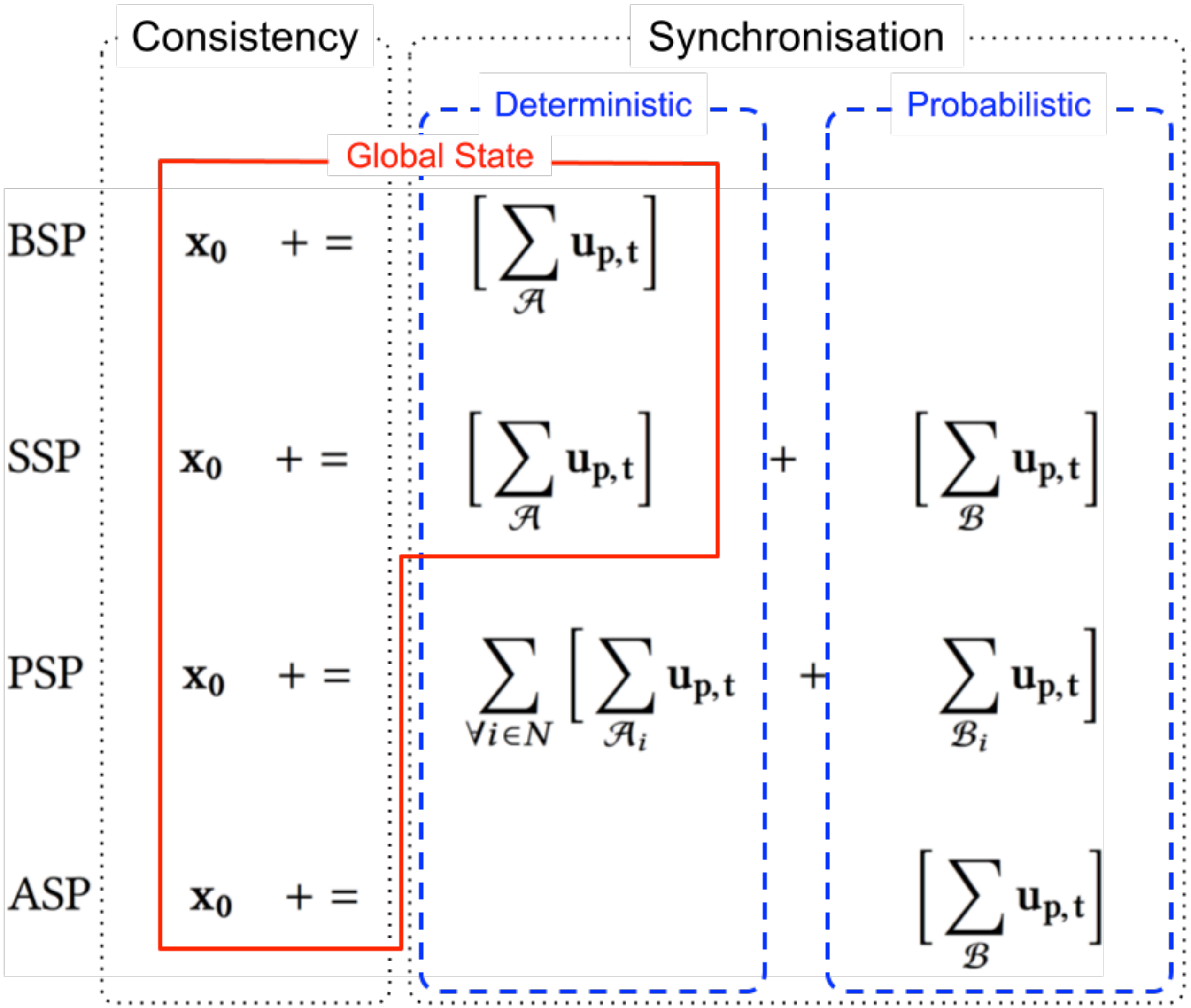
First two (BSP, SSP) are centralised; whereas last two (PSP, ASP) are distributed.

Unlike SSP, PSP decouples the consistency and the synchronisation.

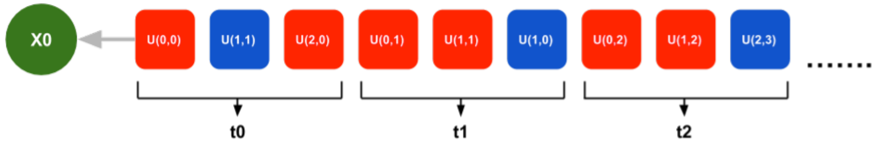
Unlike ASP, PSP does not give up synchronisation because of decoupling.

UNIVERSITY OF CAMBRIDGE

Indication on System Design



		Deterministic		Probabilistic
BSP	$\mathbf{x}_0 \; +=$	$\left[\sum_{\mathcal{A}} \mathbf{u}_{p,t}\right]$		
SSP	$\mathbf{x}_0 \; +=$	$\left[\sum_{\mathcal{A}} \mathbf{u}_{p,t}\right]$	$+$	$\left[\sum_{\mathcal{B}} \mathbf{u}_{p,t}\right]$
PSP	$\mathbf{x}_0 \; +=$	$\sum_{\forall i \in N} \left[\sum_{\mathcal{A}_i} \mathbf{u}_{p,t}\right.$	$+$	$\left.\sum_{\mathcal{B}_i} \mathbf{u}_{p,t}\right]$
ASP	$\mathbf{x}_0 \; +=$			$\left[\sum_{\mathcal{B}} \mathbf{u}_{p,t}\right]$



With decoupling, the server becomes a stream server, i.e. processes a sequence of submitted updates only, much easier to implement an efficient system.

Nodes coordination is fully distributed, mitigates bottleneck and single point failure to some extent.

Increment deployment as a higher-order function, compatible with existing data analytics frameworks.

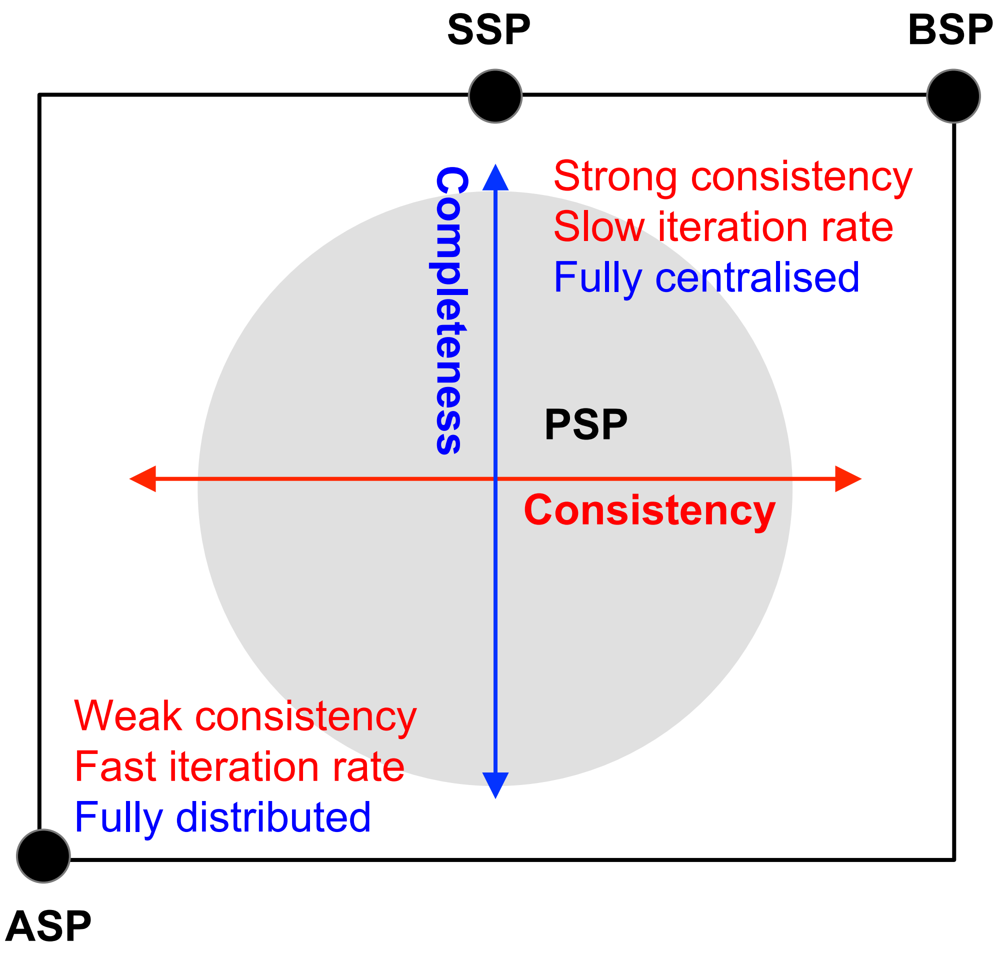
UNIVERSITY OF CAMBRIDGE

Comparison Summary

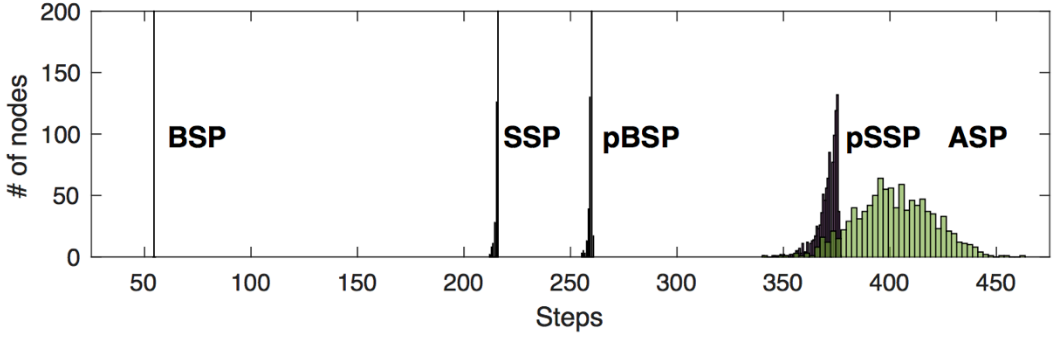
PSP vs BSP: faster iteration, faster convergence, decentralised.

PSP vs SSP : faster iteration, faster convergence, decentralised.

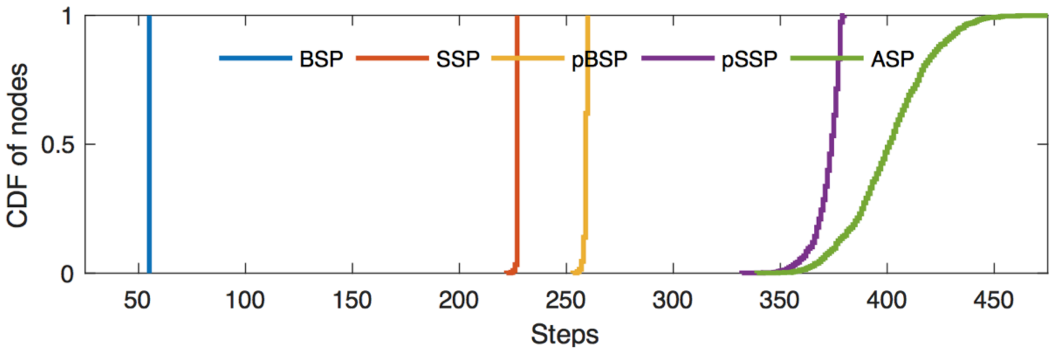
PSP vs ASP: stronger consistency with synchronisation, stronger guarantees on convergence.



Step Distribution



(a) Progress distribution in steps



(b) CDF of nodes as a function of progress. Algorithm behaviour is controlled by the single parameter sample size to mimic other solutions. No global state is maintained by any single node.

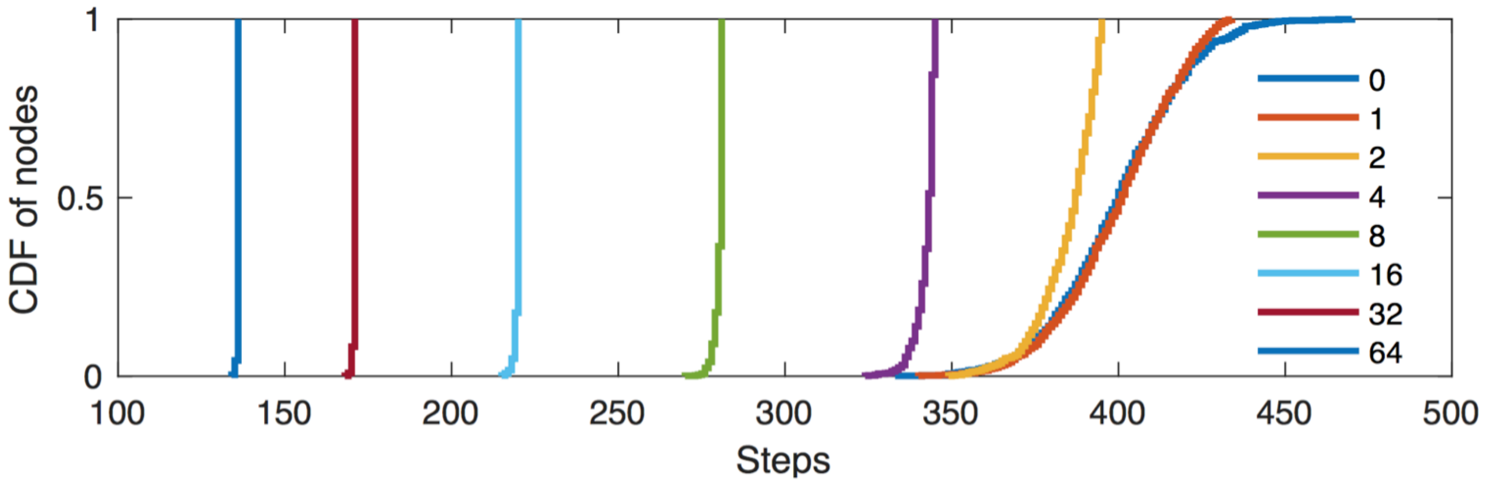
Let’s look at a system of 1000 nodes, running stochastic gradient descent algorithm for a linear regression task.

Step distribution reflects the level of coordination, or how tightly the nodes are synchronised, or degree of consistency ...

Sample size is 10, and staleness is 4. PSP trade-off reasonable consistency for much faster iteration rate.

Note the trade-off betw. the speed and spread!

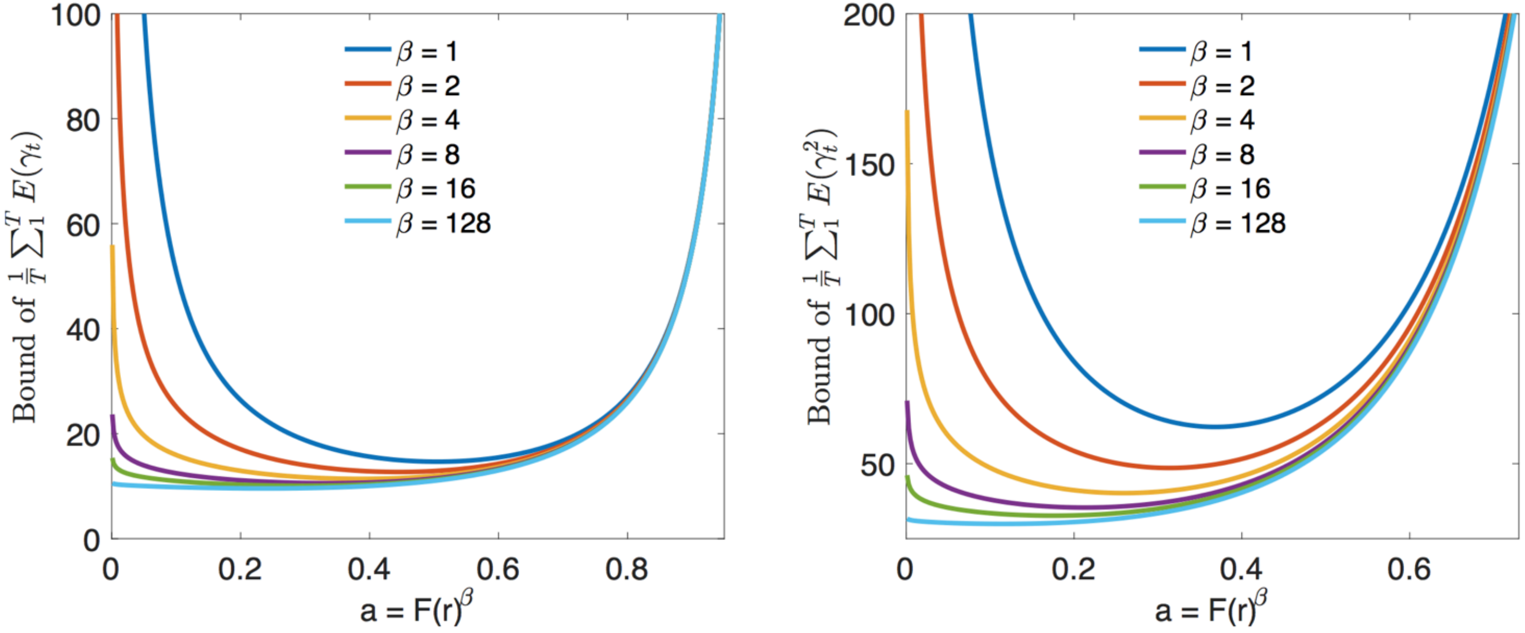
Reduce Discrepancy



CDF of step distribution at a given time in the system, we experimented with different sample size from 0 to 64. Increasing the sample size make the curves shift from right to left with decreasing spread, covering the whole spectrum from the most lenient ASP to the most strict BSP.

The smaller spread is at the price of slower iteration rate. Small sample sets seem very effective in reducing the spread.

Tighten the Bounds



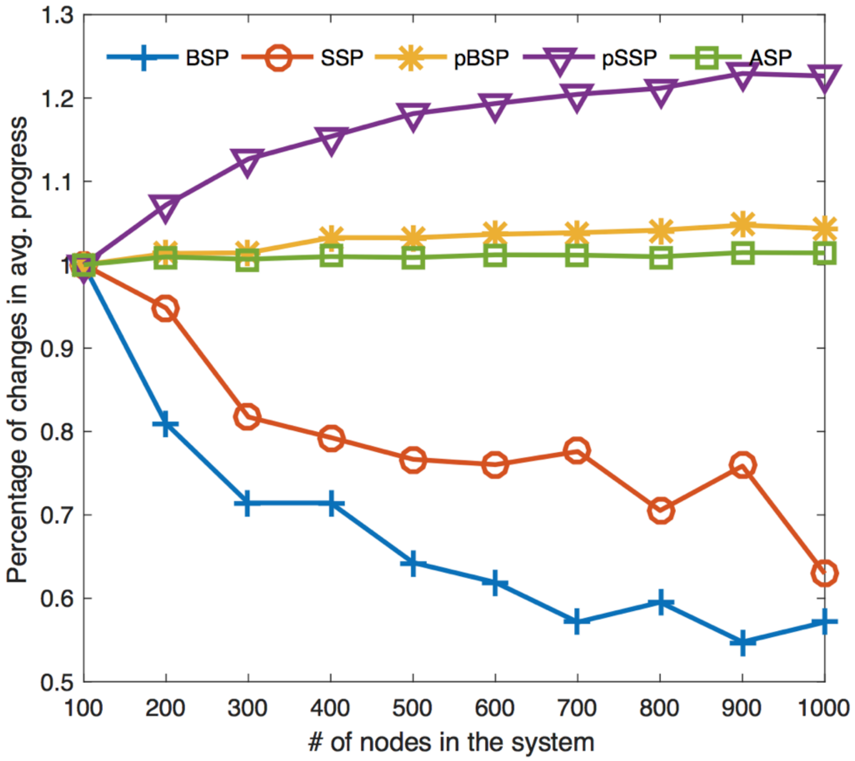
(a) Bound on means as a function of $F(r)^\beta$.

(b) Bound on variances as a function of $F(r)^\beta$

Fig. 6. Plot showing the bound on the average of the means and variances of the sampling distribution. The sampling count β is varied between 1 and 100 and marked with different line colours on the right. The staleness, r, is set to 4 with T equal to 10000.

The smaller the area below the curve is, the tighter the bound becomes, leads to stronger guarantee on convergence.

Scalability



(a) Percentage of changes in average progress as a function of systme size.

Now, we increase the system size step by step, then check how much it will degrade the performance. (Note sample size is fixed as 10.)

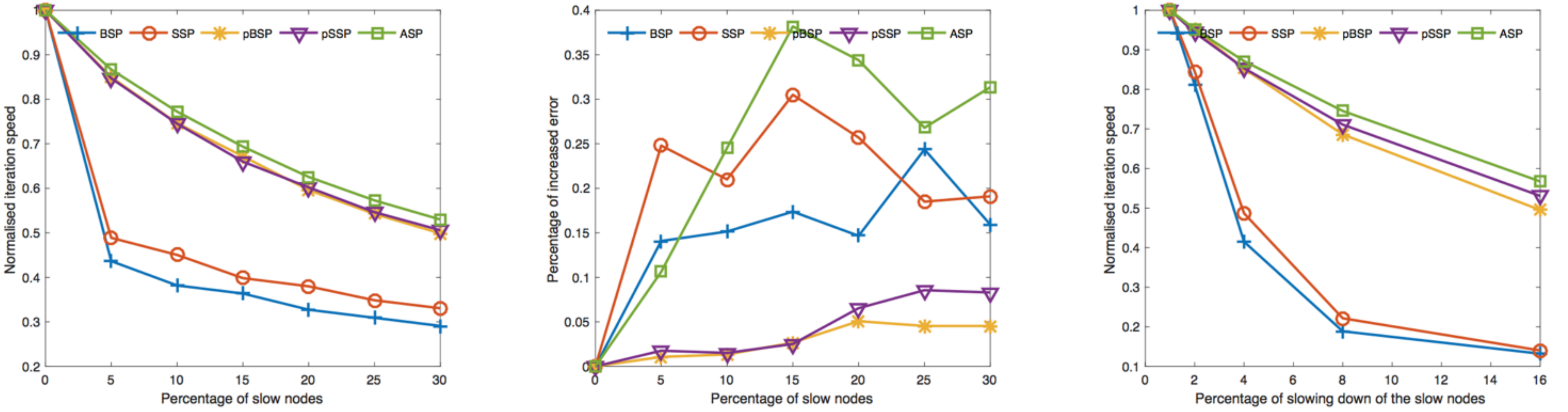
Performance degradation is measured by checking the percent of changes regarding the accuracy of regression model at a fixed timestamp in each experiment (with varying system sizes).

ASP is a straight line as expected, due to no synchronisation is need so the system is very scalable. On the other hand, BSP and SSP are both degraded.

Interestingly, PSP improves the performance. Why is that?

UNIVERSITY OF CAMBRIDGE

Robustness against Stragglers



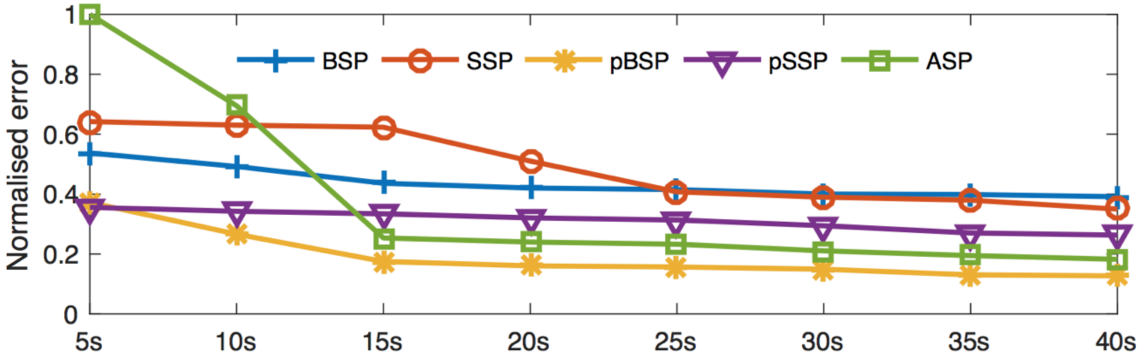
(a) Normalised average speed as a function of percentage of the slow nodes from 0% to 30%.

(b) Percentage of increased error as a function of percentage of slow nodes from 0% to 30%.

(c) Keep 5% slow nodes, increase their slowness step by step from 2x to 16x slow.

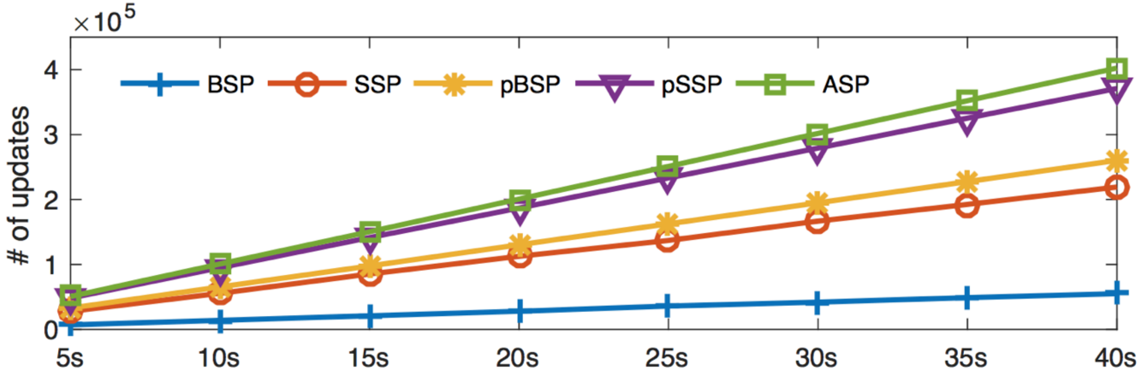
Fig. 2. Stragglers impact both system performance and accuracy of model updates. Probabilistic synchronisation control by sampling primitive is able to mitigate such impacts.

Convergence Rate



(d) Normalized error value; note that the measurements all taken at the same times (5 s, 10 s, etc.)

PSP can effectively accelerate convergence rate due to both increased iteration rate and improved consistency.



(e) Aggregated number of updates received by the server

The number of submitted updates increases due to PSP's faster iteration rate.

UNIVERSITY OF CAMBRIDGE

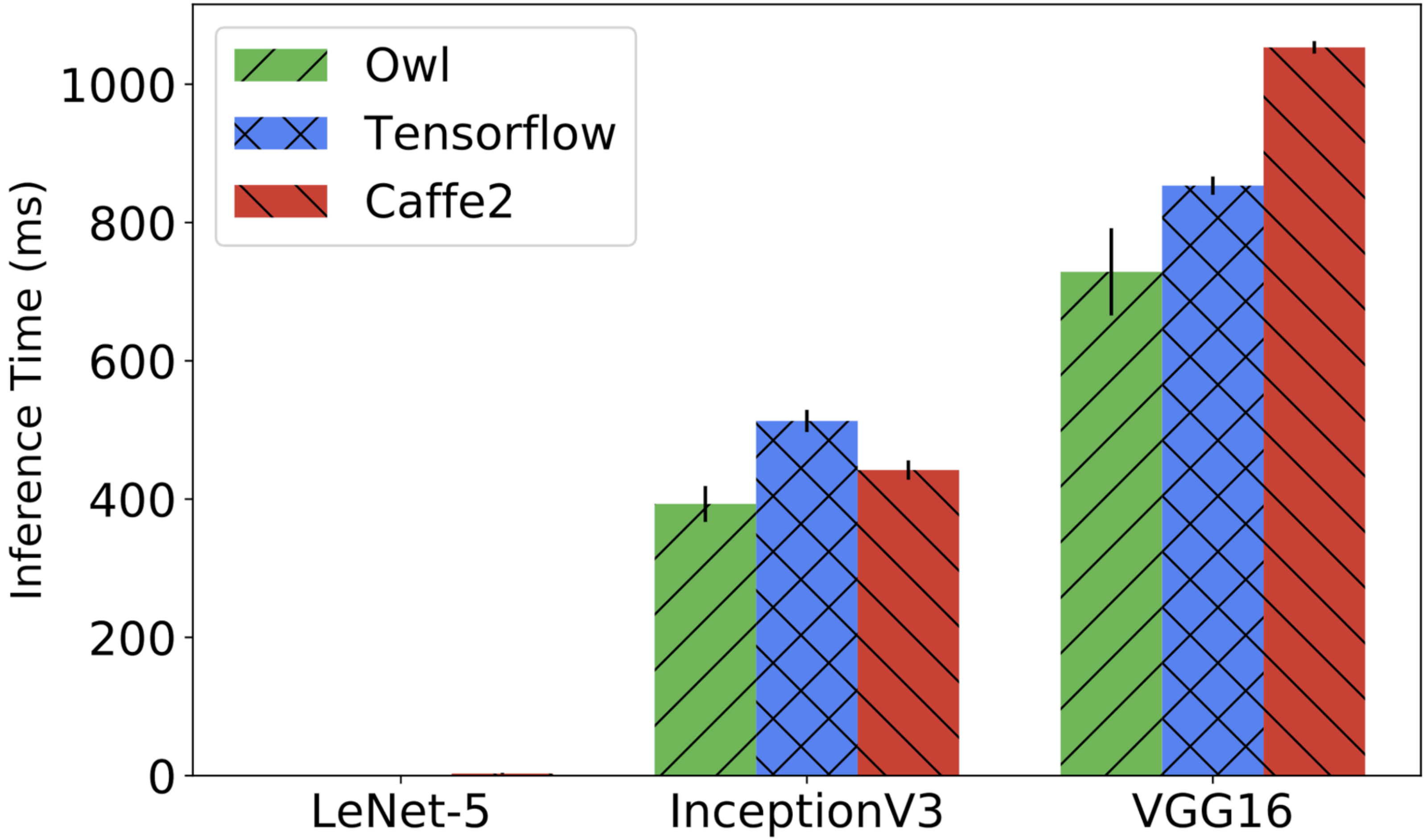
Implementation in Actor

```
type barrier =
  | ASP    (* Asynchronous Parallel *)
  | BSP    (* Bulk Synchronous Parallel *)
  | SSP    (* Stale Synchronous Parallel *)
  | PSP    (* Probabilistic Synchronous Parallel *)

val start : ?barrier:barrier -> string -> string -> unit
(** start running the model loop *)

val register_barrier : ps_barrier_typ -> unit
(** register user-defined barrier function at p2p server *)

val register_schedule : ('a, 'b, 'c) ps_schedule_typ -> unit
(** register user-defined scheduler *)

val register_pull : ('a, 'b, 'c) ps_pull_typ -> unit
(** register user-defined pull function executed at master
*)

…….
```

PSP and other synchronous parallel machines are implemented in Actor system in a very modular way.

They can be plugged into different engines (Parameter Sever, Mapreduce) with `register_barrier` function

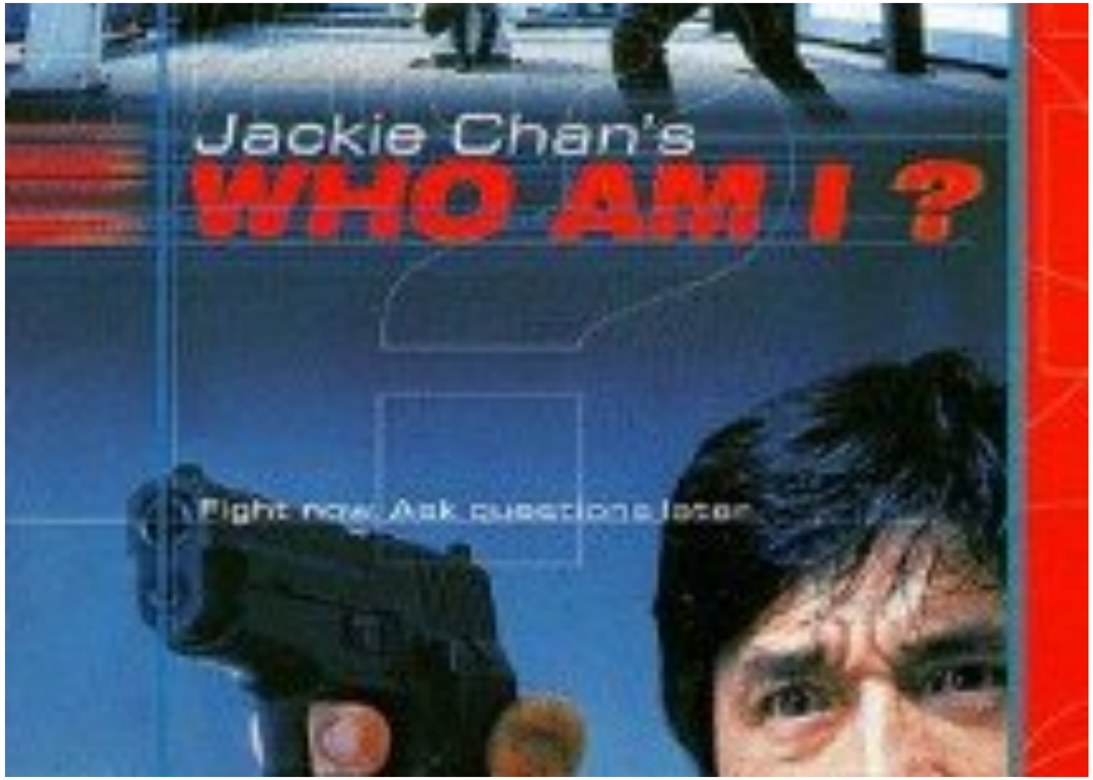
Owl’s Performance



Who Am I? & lets not speculate further ☺

Thanks to EPSRC/databox

- &Liang Wang, Cambridge



Thanks to Turing/Maru

- &Peter Pietzuch, Imperial