A unifying modeling framework for data-intensive systems

Nicolò Felicioni

POLITECNICO MILANO 1863

HONOURS PROGRAMME HP-SR in Information Technology

Introduction

Data-intensive meaning

- **Data-intensive** application: data is the primary challenge
  - Volume
  - Velocity
  - Variety
- **Compute-intensive** application: CPU is the bottleneck
  - Example: computer simulation software

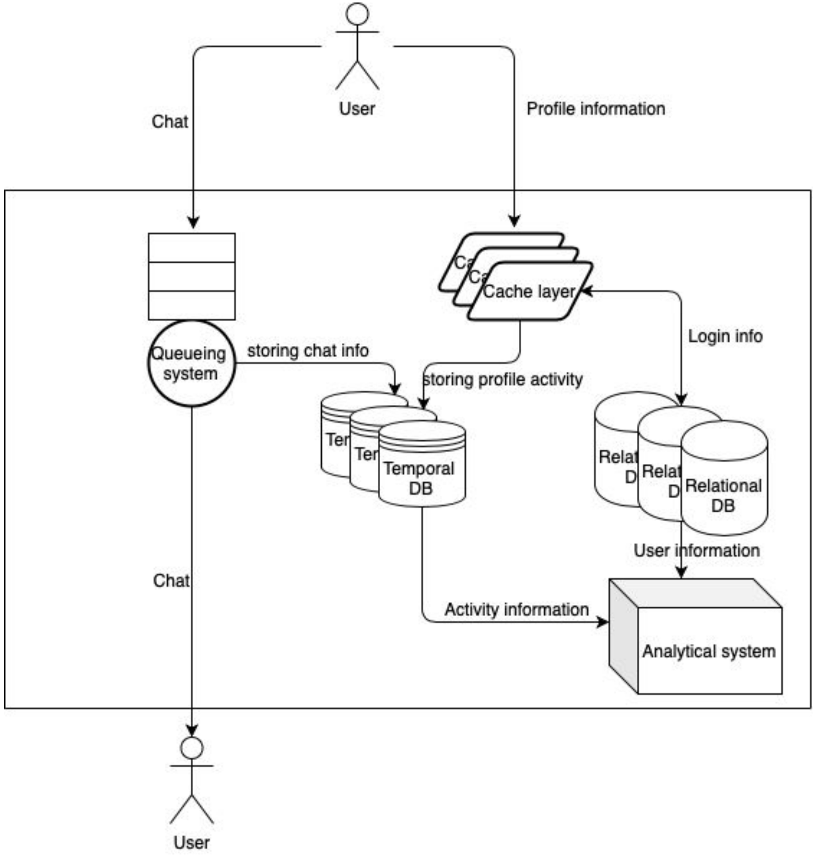
Current problems

Existing systems usually solve **specific** tasks

Data-intensive applications have **heterogeneous** requirements

Common practice: developers integrate different systems with ad-hoc manually written logic

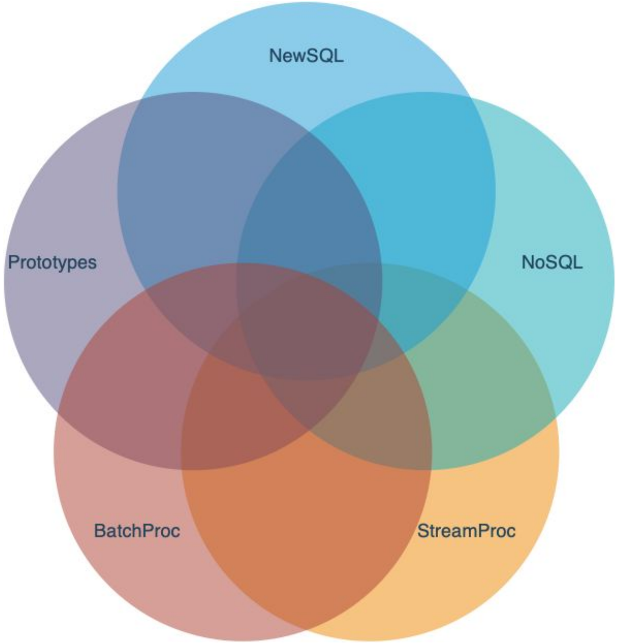
Integration needs a **deep knowledge** of each system



Motivations for a unifying model

In this scenario, a unifying model may bring several contributions:

- Select the **best** system for the problem at hand
- Know how to **configure** the systems to meet application requirements
- Understand **common design principles**
- Guide the design of a new breed of **more tightly integrated** data-intensive tools



Scope

We examine systems from various areas: database, batch processing, stream processing, etc.

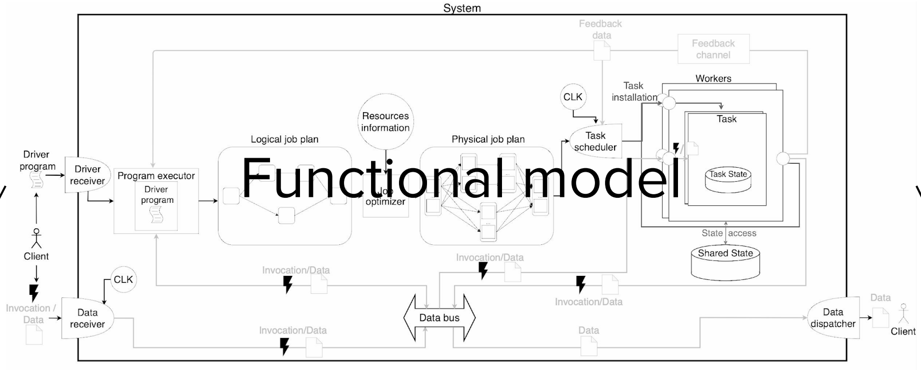
We look at the most representative systems for each area

In total, we analyzed 16 among the most relevant systems in the different domains

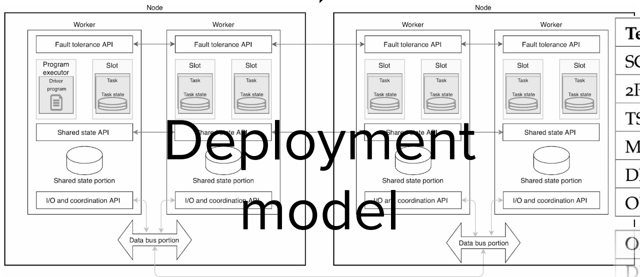
Data-intensive class		Analyzed systems
Database	**NoSQL**	Cassandra MongoDB
	NewSQL	VoltDB Calvin Spanner
	Research prototype	StreamDB ReactDB
Batch processing		MapReduce Spark
Stream processing		Flink KafkaStreams Samza Spark Streaming TSpoon
Hybrid		S-Store SnappyData

The modeling framework

The modeling framework



Functional model



Deployment model

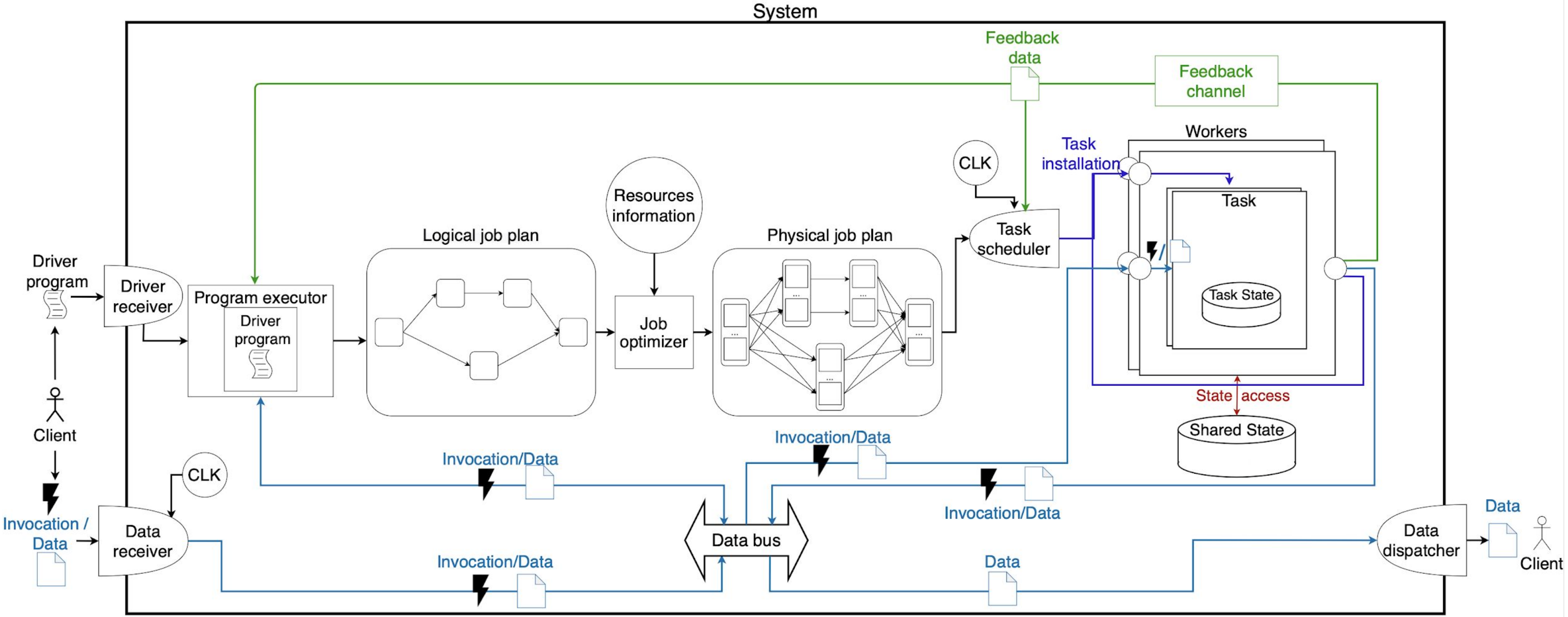
Guarantees model

Technique	Preconditions	Deadlocks	Locks/Timestamp
SCHED	Single-slot transactions	No	None
2PL	None	Yes	Locks
TS	None	No	Timestamp
MVCC	None	No	Timestamp
DET	Deterministic transactions	No	Hybrid
OLLP	No system-induced aborts	No	Hybrid

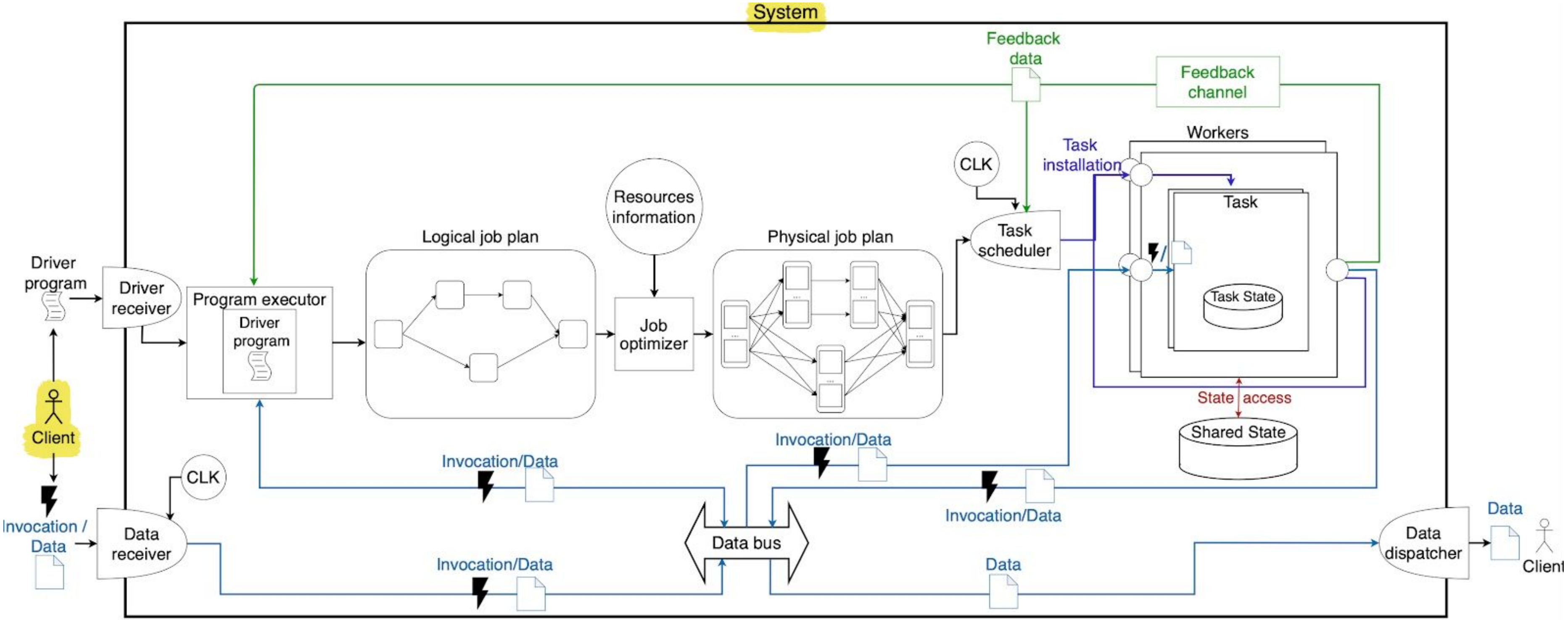
Guarantees' implementation model

Technique	Preconditions	Deadlocks	Locks/Timestamp
SCHED	Single-slot transactions	No	None
2PL	/	Yes	Locks
TS	/	No	Timestamp
MVCC	/	No	Timestamp
DET	Deterministic transactions	No	Hybrid
OLLP	No system-induced aborts	No	Hybrid

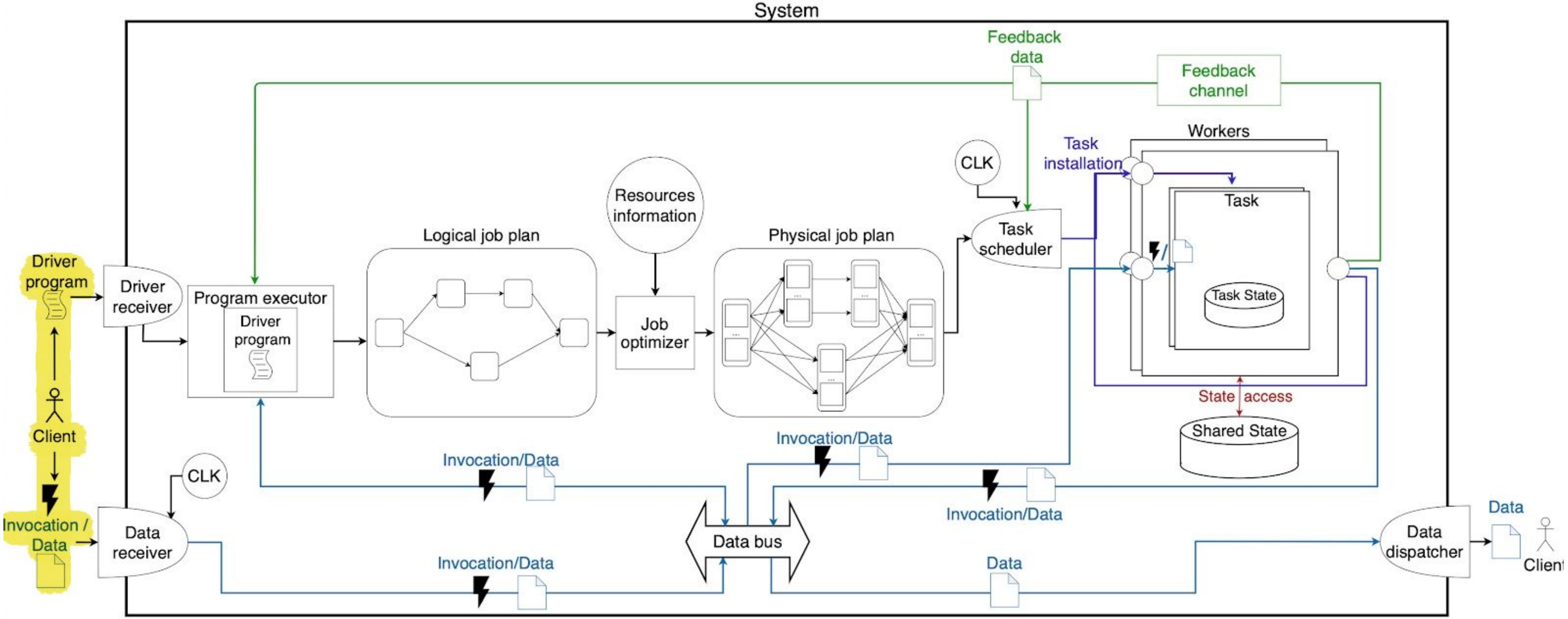
Functional model



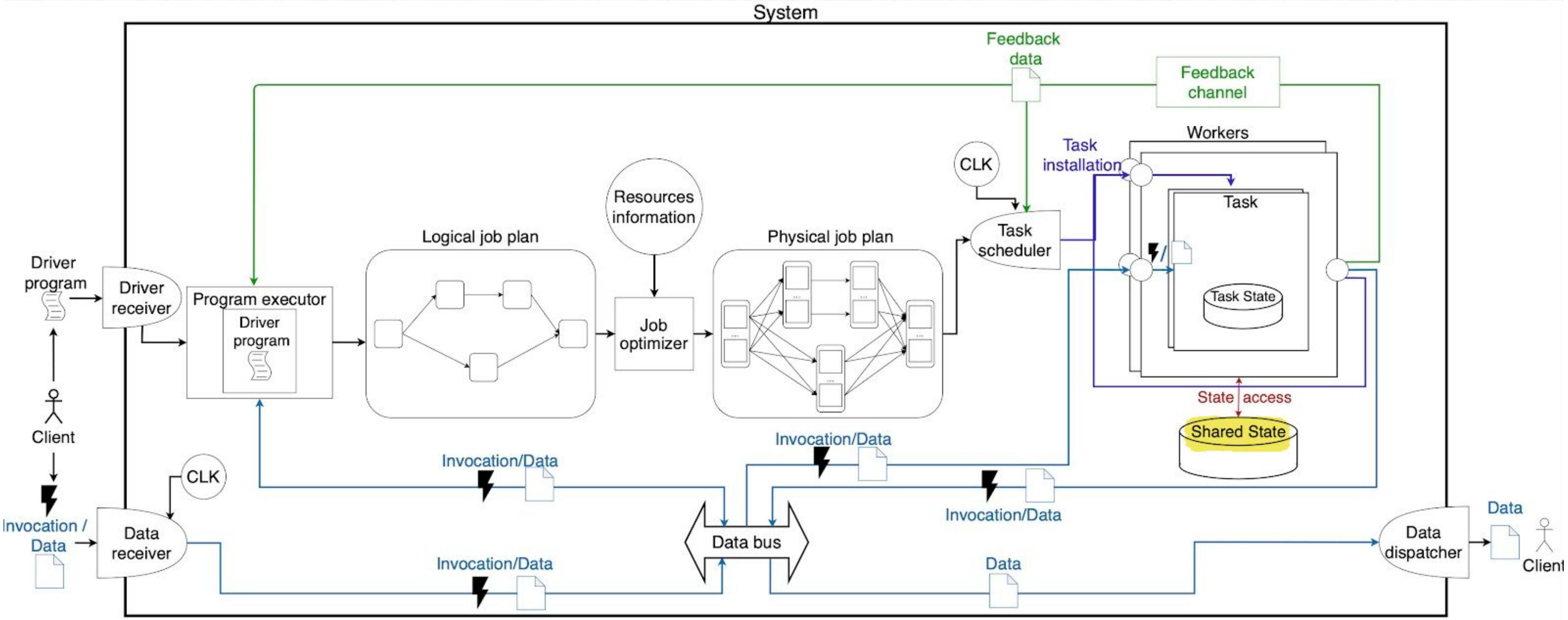
Functional model



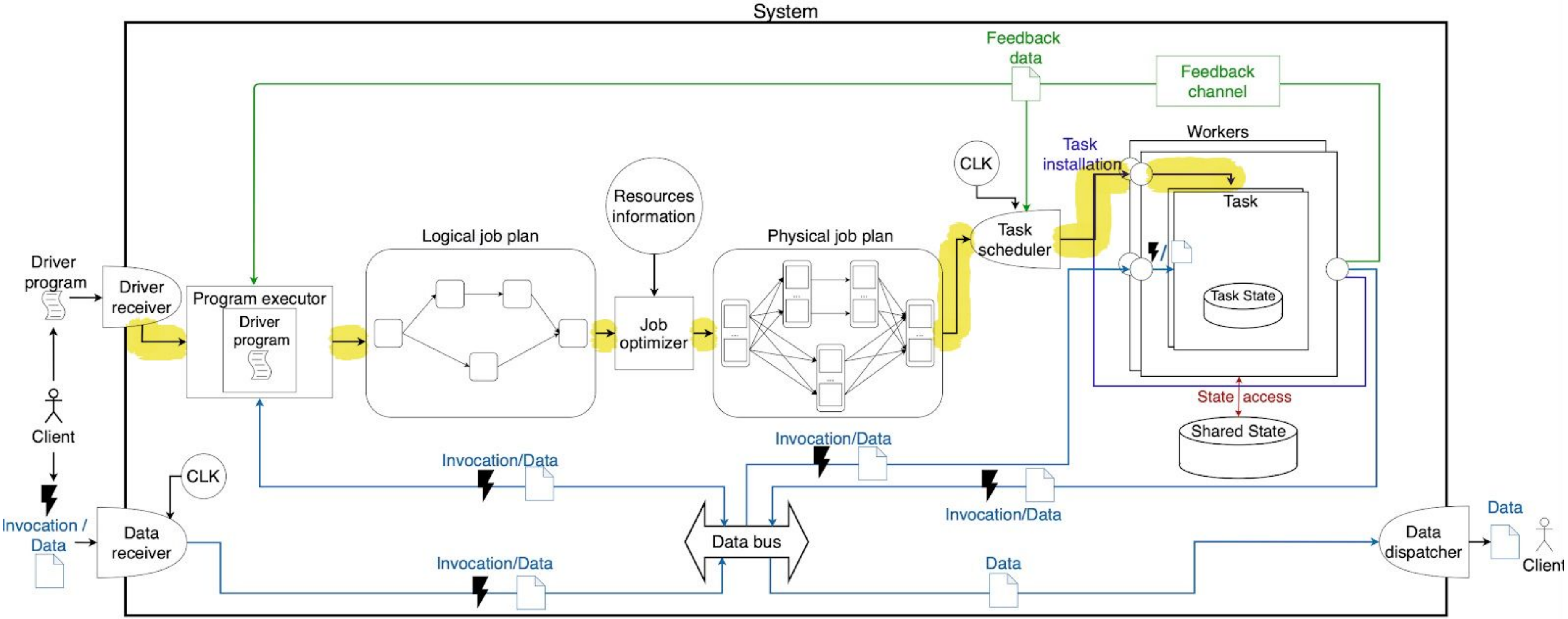
Functional model



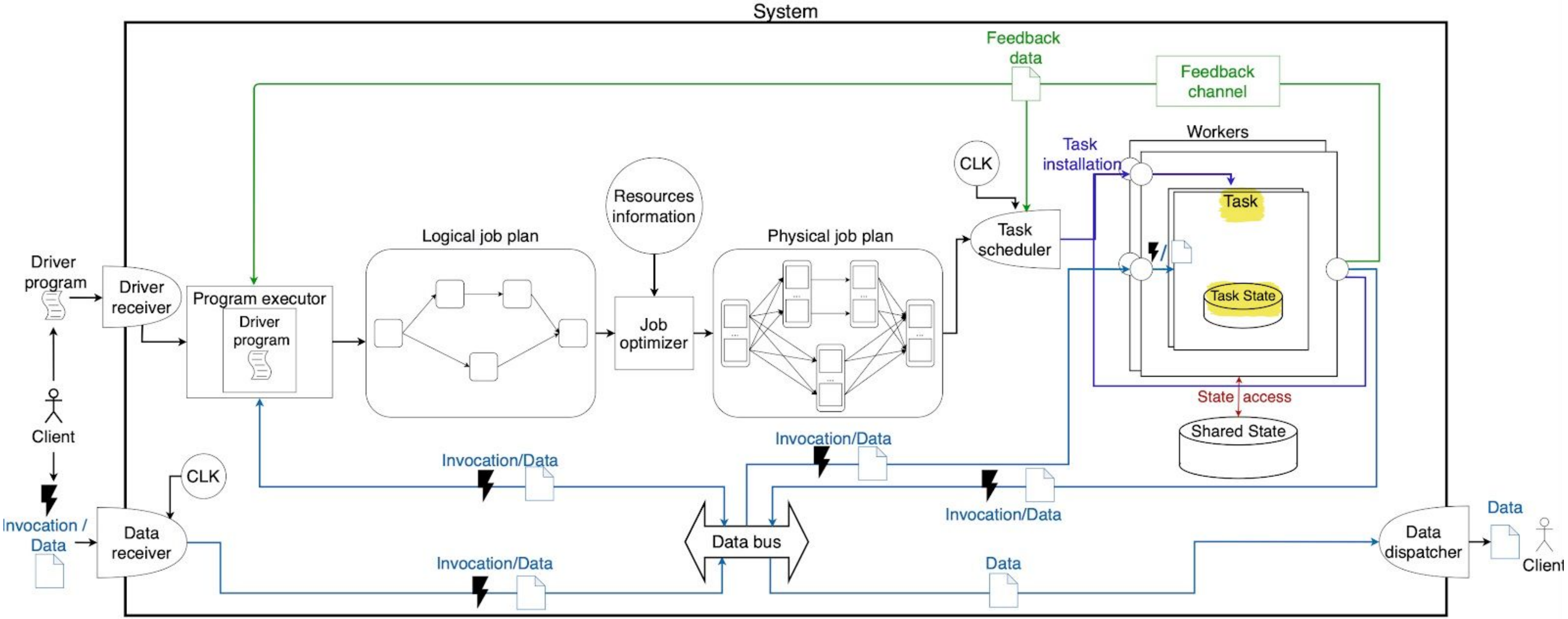
Functional model



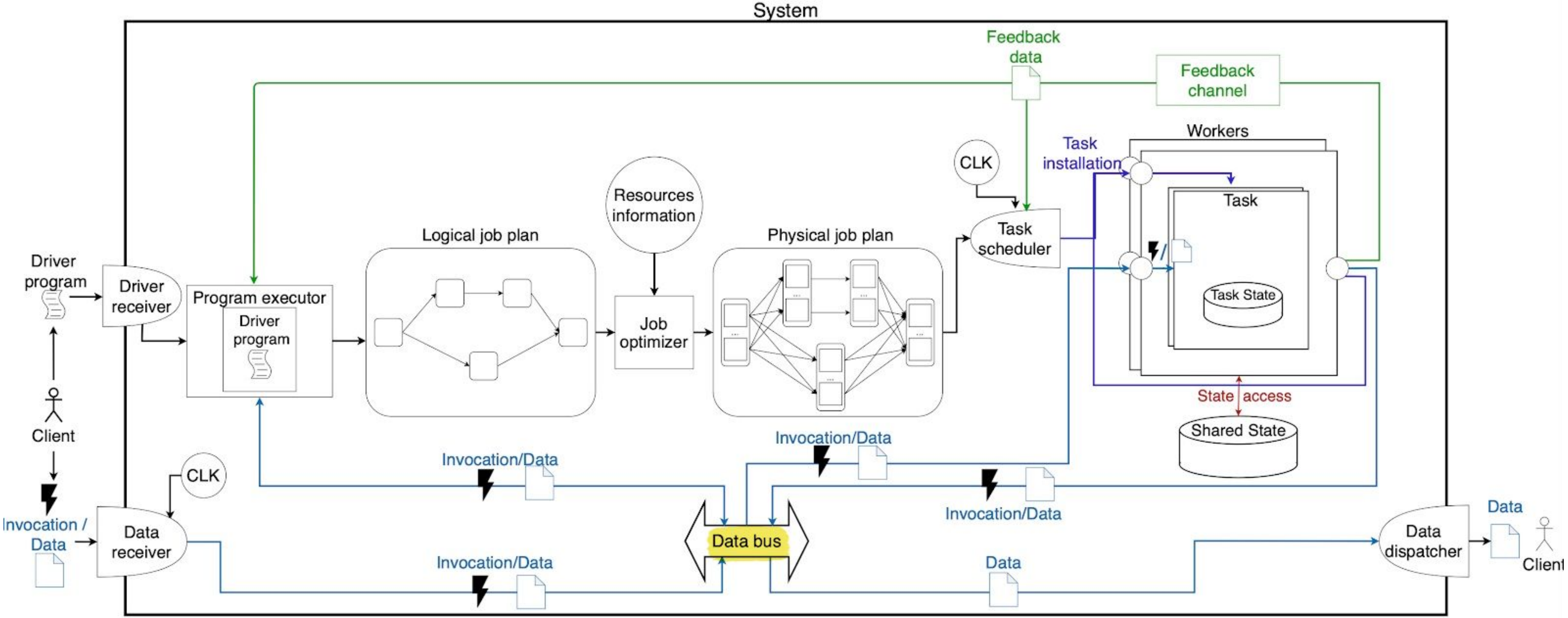
Functional model



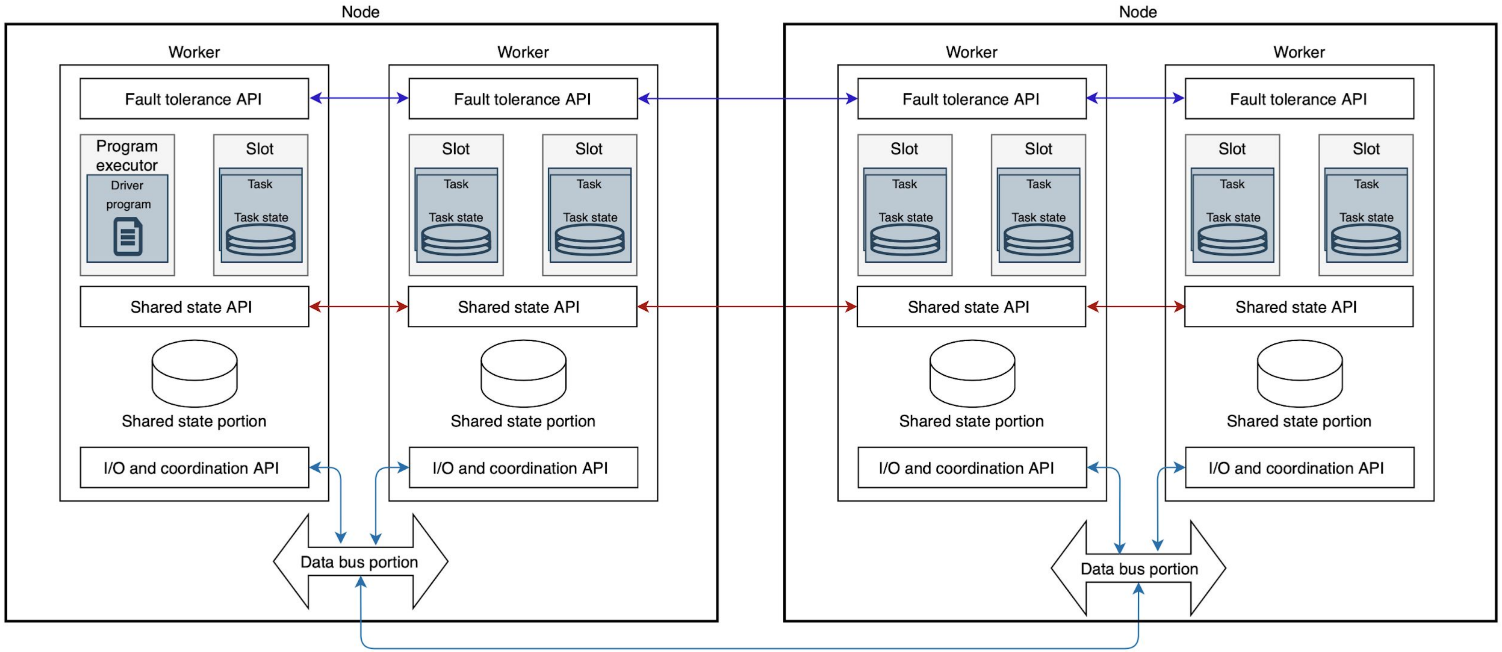
Functional model



Functional model



Deployment model



Guarantees model

We defined guarantees as **postconditions** that a client can assume being true after the computation

There are a lot of guarantees that are difficult to provide in a data-intensive environment

Synchronization (atomicity, isolation), fault tolerance, communication (delivery, order), consistency, etc.

Guarantees' implementation model

Atomicity

Technique	Preconditions	Coordination
2PC	/	Synchronous
Scheduling	No system-induced aborts	Asynchronous
	No system-induced or logic-induced aborts	Free
	Single-slot transaction	Free

Fault tolerance

Technique	Preconditions	Stable storage	Valid/Same state
WAL(+SNPSHT)	/	Disk	Same
CL(+SNPSHT)	/	Disk	Valid
	Deterministic transactions	Disk	Same
SNPSHT	Client sends data from snpsht	Disk	Valid
REPL	/	Disk (multiple nodes)	Same

Isolation

Technique	Preconditions	Deadlocks	Locks/Timestamp
SCHED	Single-slot transactions	No	None
2PL	/	Yes	Locks
TS	/	No	Timestamp
MVCC	/	No	Timestamp
DET	Deterministic transactions	No	Hybrid
OLLP	No system-induced aborts	No	Hybrid

Replication consistency

Technique	Preconditions	Level
CRDT	/	Eventual
Application code	/	Eventual
Single leader	/	Sequential
Sync. single leader	/	Linearizable
Sync. quorum based	/	Linearizable
Deterministic	Deterministic transactions	Linearizable

Discussion

Functional model

- Pull vs Push vs Periodic approach
- State management
- Almost all systems have an optimizer

		Invocations	Shared state	Task state	Job optimizer
DB	**Cassandra**	Pull	Yes	No	Yes
	MongoDB	Pull	Yes	No	Yes
	VoltDB	Pull/push	Yes	No	Yes
	Calvin	Pull	Yes	No	Yes
	Spanner	Pull	Yes	No	Yes
	StreamDB	Push	Yes	No	Yes
	ReactDB	Pull	Yes	No	No
BP	**MapReduce**	Pull	No	No	Yes
	Spark	Pull	No	No	Yes
SP	**Flink**	Push	No	Yes	Yes
	KafkaStreams	Push	No	Yes	Yes
	Samza	Push	No	Yes	Yes
	Spark Streaming	Periodic	No	Yes	Yes
	TSpoon	Push	No	Yes	Yes
Hyb	**S-Store**	Pull/push	Yes	Yes	Yes
	SnappyData	Pull/periodic	Yes	Yes	Yes

Functional model

- Pull vs Push vs Periodic approach
- State management
- Almost all systems have an optimizer

		Invocations	Shared state	Task state	Job optimizer
DB	Cassandra	Pull	Yes	No	Yes
	MongoDB	Pull	Yes	No	Yes
	VoltDB	Pull/push	Yes	No	Yes
	Calvin	Pull	Yes	No	Yes
	Spanner	Pull	Yes	No	Yes
	StreamDB	Push	Yes	No	Yes
	ReactDB	Pull	Yes	No	No
BP	MapReduce	Pull	No	No	Yes
	Spark	Pull	No	No	Yes
SP	Flink	Push	No	Yes	Yes
	KafkaStreams	Push	No	Yes	Yes
	Samza	Push	No	Yes	Yes
	Spark Streaming	Periodic	No	Yes	Yes
	TSpoon	Push	No	Yes	Yes
Hyb	S-Store	Pull/push	Yes	Yes	Yes
	SnappyData	Pull/periodic	Yes	Yes	Yes

Functional model

- Pull vs Push vs Periodic approach
- State management
- Almost all systems have an optimizer

		Invocations	Shared state	Task state	Job optimizer
DB	Cassandra	Pull	Yes	No	Yes
	MongoDB	Pull	Yes	No	Yes
	VoltDB	Pull/push	Yes	No	Yes
	Calvin	Pull	Yes	No	Yes
	Spanner	Pull	Yes	No	Yes
	StreamDB	Push	Yes	No	Yes
	ReactDB	Pull	Yes	No	No
BP	MapReduce	Pull	No	No	Yes
	Spark	Pull	No	No	Yes
SP	Flink	Push	No	Yes	Yes
	KafkaStreams	Push	No	Yes	Yes
	Samza	Push	No	Yes	Yes
	Spark Streaming	Periodic	No	Yes	Yes
	TSpoon	Push	No	Yes	Yes
Hyb	S-Store	Pull/push	Yes	Yes	Yes
	SnappyData	Pull/periodic	Yes	Yes	Yes

Functional model

- Pull vs Push vs Periodic approach
- State management
- Almost all systems have an optimizer

		Invocations	Shared state	Task state	Job optimizer
DB	Cassandra	Pull	Yes	No	Yes
	MongoDB	Pull	Yes	No	Yes
	VoltDB	Pull/push	Yes	No	Yes
	Calvin	Pull	Yes	No	Yes
	Spanner	Pull	Yes	No	Yes
	StreamDB	Push	Yes	No	Yes
	ReactDB	Pull	Yes	No	No
BP	MapReduce	Pull	No	No	Yes
	Spark	Pull	No	No	Yes
SP	Flink	Push	No	Yes	Yes
	KafkaStreams	Push	No	Yes	Yes
	Samza	Push	No	Yes	Yes
	Spark Streaming	Periodic	No	Yes	Yes
	TSpoon	Push	No	Yes	Yes
Hyb	S-Store	Pull/push	Yes	Yes	Yes
	SnappyData	Pull/periodic	Yes	Yes	Yes

Deployment model

- Almost all systems have partitioning/replication
- VoltDB has always one slot per worker
- KafkaStreams architecture is unique
- Few systems have persistent data bus

		Worker	Slot	Persistence layer	Replication	Partitioning
DB	**Cassandra**	Node	No name	No	Active, leaderless	Content
	MongoDB	Mongod	No name	No	Passive Active, single-leader	Content
	VoltDB	Site	Site	No	Active, single-leader Leaderless	Content
	Calvin	Node	Execution thread	No	Active, leaderless	Content
	Spanner	Spanserver	Not mentioned	No	Active, single-leader	Content
	StreamDB	Site	Thread	No	Active, single-leader	Content
	ReactDB	Container	Transaction executor	No	No	No
BP	**MapReduce**	Master/Worker	Not mentioned	Yes	Active, leaderless	Content
	Spark	Executor	Not mentioned	Depends on data bus	Depends on data bus	Depends on data bus
SP	**Flink**	Task manager Job manager	Task slot	No	No	Content
	KafkaStreams	Client application	Thread	Yes	Passive	Content
	Samza	Container	Thread	Yes	Passive	Content
	Spark Streaming	Executor	No name	Depends on data bus	Data replication	Content
	TSpoon	Task manager Job manager	Task slot	No	No	Content
Hyb	**S-Store**	Site	Site	No	Active, single-leader Leaderless	Content
	SnappyData	Executor	Core	No	Active, single-leader	Content

Deployment model

- Almost all systems have partitioning/replication
- VoltDB has always one slot per worker
- KafkaStreams architecture is unique
- Few systems have persistent data bus

		Worker	Slot	Persistence layer	Replication	Partitioning
DB	**Cassandra**	Node	No name	No	Active, leaderless	Content
	MongoDB	Mongod	No name	No	Passive Active, single-leader	Content
	VoltDB	Site	Site	No	Active, single-leader Leaderless	Content
	Calvin	Node	Execution thread	No	Active, leaderless	Content
	Spanner	Spanserver	Not mentioned	No	Active, single-leader	Content
	StreamDB	Site	Thread	No	Active, single-leader	Content
	ReactDB	Container	Transaction executor	No	No	No
BP	**MapReduce**	Master/Worker	Not mentioned	Yes	Active, leaderless	Content
	Spark	Executor	Not mentioned	Depends on data bus	Depends on data bus	Depends on data bus
SP	**Flink**	Task manager Job manager	Task slot	No	No	Content
	KafkaStreams	Client application	Thread	Yes	Passive	Content
	Samza	Container	Thread	Yes	Passive	Content
	Spark Streaming	Executor	No name	Depends on data bus	Data replication	Content
	TSpoon	Task manager Job manager	Task slot	No	No	Content
Hyb	**S-Store**	Site	Site	No	Active, single-leader Leaderless	Content
	SnappyData	Executor	Core	No	Active, single-leader	Content

Deployment model

- Almost all systems have partitioning/replication
- VoltDB has always one slot per worker
- KafkaStreams architecture is unique
- Few systems have persistent data bus

		Worker	Slot	Persistence layer	Replication	Partitioning
DB	Cassandra	Node	No name	No	Active, leaderless	Content
	MongoDB	Mongod	No name	No	Passive Active, single-leader	Content
	VoltDB	Site	Site	No	Active, single-leader Leaderless	Content
	Calvin	Node	Execution thread	No	Active, leaderless	Content
	Spanner	Spanserver	Not mentioned	No	Active, single-leader	Content
	StreamDB	Site	Thread	No	Active, single-leader	Content
	ReactDB	Container	Transaction executor	No	No	No
BP	MapReduce	Master/Worker	Not mentioned	Yes	Active, leaderless	Content
	Spark	Executor	Not mentioned	Depends on data bus	Depends on data bus	Depends on data bus
SP	Flink	Task manager Job manager	Task slot	No	No	Content
	KafkaStreams	Client application	Thread	Yes	Passive	Content
	Samza	Container	Thread	Yes	Passive	Content
	Spark Streaming	Executor	No name	Depends on data bus	Data replication	Content
	TSpoon	Task manager Job manager	Task slot	No	No	Content
Hyb	S-Store	Site	Site	No	Active, single-leader Leaderless	Content
	SnappyData	Executor	Core	No	Active, single-leader	Content

Deployment model

- Almost all systems have partitioning/replication
- VoltDB has always one slot per worker
- KafkaStreams architecture is unique
- Few systems have persistent data bus

		Worker	Slot	Persistence layer	Replication	Partitioning
DB	Cassandra	Node	No name	No	Active, leaderless	Content
	MongoDB	Mongod	No name	No	Passive Active, single-leader	Content
	VoltDB	Site	Site	No	Active, single-leader Leaderless	Content
	Calvin	Node	Execution thread	No	Active, leaderless	Content
	Spanner	Spanserver	Not mentioned	No	Active, single-leader	Content
	StreamDB	Site	Thread	No	Active, single-leader	Content
	ReactDB	Container	Transaction executor	No	No	No
BP	MapReduce	Master/Worker	Not mentioned	Yes	Active, leaderless	Content
	Spark	Executor	Not mentioned	Depends on data bus	Depends on data bus	Depends on data bus
SP	Flink	Task manager Job manager	Task slot	No	No	Content
	KafkaStreams	Client application	Thread	Yes	Passive	Content
	Samza	Container	Thread	Yes	Passive	Content
	Spark Streaming	Executor	No name	Depends on data bus	Data replication	Content
	TSpoon	Task manager Job manager	Task slot	No	No	Content
Hyb	S-Store	Site	Site	No	Active, single-leader Leaderless	Content
	SnappyData	Executor	Core	No	Active, single-leader	Content

Deployment model

- Almost all systems have partitioning/replication
- VoltDB has always one slot per worker

- KafkaStreams architecture is unique
- Few systems have persistent data bus

		Worker	Slot	Persistence layer	Replication	Partitioning
DB	Cassandra	Node	No name	No	Active, leaderless	Content
	MongoDB	Mongod	No name	No	Passive Active, single-leader	Content
	VoltDB	Site	Site	No	Active, single-leader Leaderless	Content
	Calvin	Node	Execution thread	No	Active, leaderless	Content
	Spanner	Spanserver	Not mentioned	No	Active, single-leader	Content
	StreamDB	Site	Thread	No	Active, single-leader	Content
	ReactDB	Container	Transaction executor	No	No	No
BP	MapReduce	Master/Worker	Not mentioned	Yes	Active, leaderless	Content
	Spark	Executor	Not mentioned	Depends on data bus	Depends on data bus	Depends on data bus
SP	Flink	Task manager Job manager	Task slot	No	No	Content
	KafkaStreams	Client application	Thread	Yes	Passive	Content
	Samza	Container	Thread	Yes	Passive	Content
	Spark Streaming	Executor	No name	Depends on data bus	Data replication	Content
	TSpoon	Task manager Job manager	Task slot	No	No	Content
Hyb	S-Store	Site	Site	No	Active, single-leader Leaderless	Content
	SnappyData	Executor	Core	No	Active, single-leader	Content

Guarantees and implementation model

- In general, recurring implementations used for several systems
- Among SPs, only TSpoon provides explicit support for transactions
- Some DBMSs want to provide transactional guarantees, no matter which are the preconditions

		Atomicity		Isolation	
		Impl.	**Precond.**	**Impl.**	**Precond.**
DB	**Cassandra**	SCHED	Single-partition txn	No	/
	MongoDB	SCHED	Single-document write	MVCC	/
		2PC	/		
	VoltDB	SCHED	Single-slot txn/one-shot RO txn	DET	Det. txns
		2PC	/		
	Calvin	SCHED	No system-induced aborts (async coord.)	DET	Det. txns
			No aborts (coord. free)	OLLP	Non-statically analyzable
	Spanner	2PC	/	MVCC	RO txns
				S2PL	/
	StreamDB	SCHED	Single-slot txn	TS	/
	ReactDB	SCHED	Single-container txn	OCC	/
		2PC	/		
SP	**Flink**	SCHED	Single-slot txn	SCHED	Single-slot txn
	KafkaStreams	2PC	Single-slot txn	SCHED	Single-slot txn
	Samza	SCHED	Single-slot txn	SCHED	Single-slot txn
	Spark Streaming	SCHED	Single-slot txn	SCHED	Single-slot txn
	TSpoon	2PC	/	2PL	/
				TS	/
Hyb	**S-Store**	SCHED	Single-slot txn One-shot RO txn	DET	Det. txns
		2PC	/		
	SnappyData	2PC	/	2PL	/

Guarantees and implementation model

- In general, recurring implementations used for several systems
- Among SPs, only TSpoon provides explicit support for transactions
- Some DBMSs want to provide transactional guarantees, no matter which are the preconditions

		Atomicity		Isolation	
		Impl.	**Precond.**	**Impl.**	**Precond.**
DB	**Cassandra**	SCHED	Single-partition txn	No	/
	MongoDB	SCHED	Single-document write	MVCC	/
		2PC	/		
	VoltDB	SCHED	Single-slot txn/one-shot RO txn	DET	Det. txns
		2PC	/		
	Calvin	SCHED	No system-induced aborts (async coord.)	DET	Det. txns
			No aborts (coord. free)	OLLP	Non-statically analyzable
	Spanner	2PC	/	MVCC	RO txns
				S2PL	/
	StreamDB	SCHED	Single-slot txn	TS	/
	ReactDB	SCHED	Single-container txn	OCC	/
		2PC	/		
SP	**Flink**	SCHED	Single-slot txn	SCHED	Single-slot txn
	KafkaStreams	2PC	Single-slot txn	SCHED	Single-slot txn
	Samza	SCHED	Single-slot txn	SCHED	Single-slot txn
	Spark Streaming	SCHED	Single-slot txn	SCHED	Single-slot txn
	TSpoon	2PC	/	2PL	/
				TS	/
Hyb	**S-Store**	SCHED	Single-slot txn One-shot RO txn	DET	Det. txns
		2PC	/		
	SnappyData	2PC	/	2PL	/

Guarantees and implementation model

- In general, recurring implementations used for several systems
- Among SPs, only TSpoon provides explicit support for transactions
- Some DBMSs want to provide transactional guarantees, no matter which are the preconditions

		Atomicity		Isolation	
		Impl.	**Precond.**	**Impl.**	**Precond.**
DB	**Cassandra**	SCHED	Single-partition txn	No	/
	MongoDB	SCHED	Single-document write	MVCC	/
		2PC	/		
	VoltDB	SCHED	Single-slot txn/one-shot RO txn	DET	Det. txns
		2PC	/		
	Calvin	SCHED	No system-induced aborts (async coord.)	DET	Det. txns
			No aborts (coord. free)	OLLP	Non-statically analyzable
	Spanner	2PC	/	MVCC	RO txns
				S2PL	/
	StreamDB	SCHED	Single-slot txn	TS	/
	ReactDB	SCHED	Single-container txn	OCC	/
		2PC	/		
SP	**Flink**	SCHED	Single-slot txn	SCHED	Single-slot txn
	KafkaStreams	2PC	Single-slot txn	SCHED	Single-slot txn
	Samza	SCHED	Single-slot txn	SCHED	Single-slot txn
	Spark Streaming	SCHED	Single-slot txn	SCHED	Single-slot txn
	TSpoon	2PC	/	2PL	/
				TS	/
Hyb	**S-Store**	SCHED	Single-slot txn One-shot RO txn	DET	Det. txns
		2PC	/		
	SnappyData	2PC	/	2PL	/

Guarantees and implementation model

- In general, recurring implementations used for several systems
- Among SPs, only TSpoon provides explicit support for transactions
- Some DBMSs want to provide transactional guarantees, no matter which are the preconditions

		Atomicity		Isolation	
		Impl.	**Precond.**	**Impl.**	**Precond.**
DB	**Cassandra**	SCHED	Single-partition txn	No	/
	MongoDB	SCHED	Single-document write	MVCC	/
		2PC	/		
	VoltDB	SCHED	Single-slot txn/one-shot RO txn	DET	Det. txns
		2PC	/		
	Calvin	SCHED	No system-induced aborts (async coord.)	DET	Det. txns
			No aborts (coord. free)	OLLP	Non-statically analyzable
	Spanner	2PC	/	MVCC	RO txns
				S2PL	/
	StreamDB	SCHED	Single-slot txn	TS	/
	ReactDB	SCHED	Single-container txn	OCC	/
		2PC	/		
SP	**Flink**	SCHED	Single-slot txn	SCHED	Single-slot txn
	KafkaStreams	2PC	Single-slot txn	SCHED	Single-slot txn
	Samza	SCHED	Single-slot txn	SCHED	Single-slot txn
	Spark Streaming	SCHED	Single-slot txn	SCHED	Single-slot txn
	TSpoon	2PC	/	2PL	/
				TS	/
Hyb	**S-Store**	SCHED	Single-slot txn One-shot RO txn	DET	Det. txns
		2PC	/		
	SnappyData	2PC	/	2PL	/

Guarantees and implementation model

- Fault tolerance provided almost by every considered system (with similar implementations)
- Ordering guarantees are more relevant in the streaming world

		Fault tolerance		**Delivery**			**Order**	
		Impl.	**Precond.**	**Level**	**Impl.**	**Precond.**	**Impl.**	**Precond.**
DB	**Cassandra**	CL+SNPSHT	/	No	/	/	No	/
		REPL	/					
	MongoDB	WAL	/	EOS	Atomicity protocols	/	No	/
	VoltDB	CL+SNPSHT	/	EOS	Atomicity protocols	/	No	/
		REPL	/					
	Calvin	CL + SNPSHT	/	EOS	Atomicity protocols	/	No	/
		REPL	/					
	Spanner	WAL	/	EOS	Atomicity protocols	/	No	/
		REPL	/					
	StreamDB	No	/	NA	NA	NA	NA	NA
	ReactDB	No	/	No	/	/	No	/
BP	**MapReduce**	No state	/	EOS	Re-execution	/	No	/
	Spark	No state	/	EOS	Re-execution	/	No	/
SP	**Flink**	SNPSHT	Client sends missing data	ALOS	SNPSHT	/	Watermarks	/
		SNPSHT after every exec.	/	EOS	SNPSHT after every exec.	/		
	KafkaStreams	CL+SNPSHT	/	EOS	2PC	/	No	/
		REPL	/					
	Samza	CL+SNPSHT	/	EOS	FT and idempotence	/	Retraction	/
		REPL	/					
	Spark Streaming	SNPSHT	Client sends missing data	ALOS	Acknowledge source	Source resend data / WAL	Batching	/
	TSpoon	WAL	/	EOS	2PC	/	Watermarks	/
Hyb.	**S-Store**	CL+SNPSHT	/	EOS	2PC	/	TS	/
		REPL	/					
	SnappyData	REPL (in-memory)	/	EOS	2PC	/	No	/

Guarantees and implementation model

- Fault tolerance provided almost by every considered system (with similar implementations)
- Ordering guarantees are more relevant in the streaming world

		Fault tolerance		**Delivery**			**Order**	
		Impl.	**Precond.**	**Level**	**Impl.**	**Precond.**	**Impl.**	**Precond.**
DB	**Cassandra**	CL+SNPSHT	/	No	/	/	No	/
		REPL	/					
	MongoDB	WAL	/	EOS	Atomicity protocols	/	No	/
	VoltDB	CL+SNPSHT	/	EOS	Atomicity protocols	/	No	/
		REPL	/					
	Calvin	CL + SNPSHT	/	EOS	Atomicity protocols	/	No	/
		REPL	/					
	Spanner	WAL	/	EOS	Atomicity protocols	/	No	/
		REPL	/					
	StreamDB	No	/	NA	NA	NA	NA	NA
	ReactDB	No	/	No	/	/	No	/
BP	**MapReduce**	No state	/	EOS	Re-execution	/	No	/
	Spark	No state	/	EOS	Re-execution	/	No	/
SP	**Flink**	SNPSHT	Client sends missing data	ALOS	SNPSHT	/	Watermarks	/
		SNPSHT after every exec.	/	EOS	SNPSHT after every exec.	/		
	KafkaStreams	CL+SNPSHT	/	EOS	2PC	/	No	/
		REPL	/					
	Samza	CL+SNPSHT	/	EOS	FT and idempotence	/	Retraction	/
		REPL	/					
	Spark Streaming	SNPSHT	Client sends missing data	ALOS	Acknowledge source	Source resend data / WAL	Batching	/
	TSpoon	WAL	/	EOS	2PC	/	Watermarks	/
Hyb.	**S-Store**	CL+SNPSHT	/	EOS	2PC	/	TS	/
		REPL	/					
	SnappyData	REPL (in-memory)	/	EOS	2PC	/	No	/

Guarantees and implementation model

- Fault tolerance provided almost by every considered system (with similar implementations)
- Ordering guarantees are more relevant in the streaming world

		Fault tolerance		Delivery			Order	
		Impl.	**Precond.**	**Level**	**Impl.**	**Precond.**	**Impl.**	**Precond.**
DB	**Cassandra**	CL+SNPSHT	/	No	/	/	No	/
		REPL	/					
	MongoDB	WAL	/	EOS	Atomicity protocols	/	No	/
	VoltDB	CL+SNPSHT	/	EOS	Atomicity protocols	/	No	/
		REPL	/					
	Calvin	CL + SNPSHT	/	EOS	Atomicity protocols	/	No	/
		REPL	/					
	Spanner	WAL	/	EOS	Atomicity protocols	/	No	/
		REPL	/					
	StreamDB	No	/	NA	NA	NA	NA	NA
	ReactDB	No	/	No	/	/	No	/
BP	**MapReduce**	No state	/	EOS	Re-execution	/	No	/
	Spark	No state	/	EOS	Re-execution	/	No	/
SP	**Flink**	SNPSHT	Client sends missing data	ALOS	SNPSHT	/	Watermarks	/
		SNPSHT after every exec.	/	EOS	SNPSHT after every exec.	/		
	KafkaStreams	CL+SNPSHT	/	EOS	2PC	/	No	/
		REPL	/					
	Samza	CL+SNPSHT	/	EOS	FT and idempotence	/	Retraction	/
		REPL	/					
	Spark Streaming	SNPSHT	Client sends missing data	ALOS	Acknowledge source	Source resend data / WAL	Batching	/
	TSpoon	WAL	/	EOS	2PC	/	Watermarks	/
Hyb.	**S-Store**	CL+SNPSHT	/	EOS	2PC	/	TS	/
		REPL	/					
	SnappyData	REPL (in-memory)	/	EOS	2PC	/	No	/

Conclusion

Conclusion

We built a modeling framework, introducing a new unbiased vocabulary

The modeling framework was used to identify overlaps and differences of data-intensive systems

We validated our framework through an analysis of the most relevant systems

Several research areas considered: databases, stream processing, batch processing, hybrids, and research prototypes

Future work

1. Expanding the taxonomy with other systems
2. Extending the framework with other models (e.g., physical)
3. Using the modeling framework for designing data-intensive systems

Thanks for the attention