Streaming Semantics for the Gannet Heterogeneous Multicore System-on-Chip

Wim Vanderbauwhede
Department of Computing Science
University of Glasgow

- Heterogeneous Multicore Systems-on-Chip
- The Gannet system
- The Gannet language
- An operational semantics for the Gannet machine
- Streaming data processing

Heterogeneous Multicore SoC

- Advances in integrated circuit technology and customer demands lead to increasing integration: entire systems on a single chip (**SoC**)
- Traditional system architecture (CPU, memory, peripherals connected over shared bus) can't scale
  - Synchronisation over large distances is impossible
  - Shared resource is performance bottleneck
- **Multicore** systems with **on-chip networks** provide a solution
  - globally asynchronous/locally synchronous
  - flexible connectivity
  - parallel processing

Heterogeneous Multicore SoC

- Evolution:

  single processor core (e.g. Pentium IV, PowerPC G4)$\longrightarrow$
  multiple cores with shared memory (e.g. Core Duo)$\longrightarrow$
  multiple cores with local memory (e.g. Larrabee)$\longrightarrow$
  multiple heterogeneous cores with local memory (e.g. Cell)

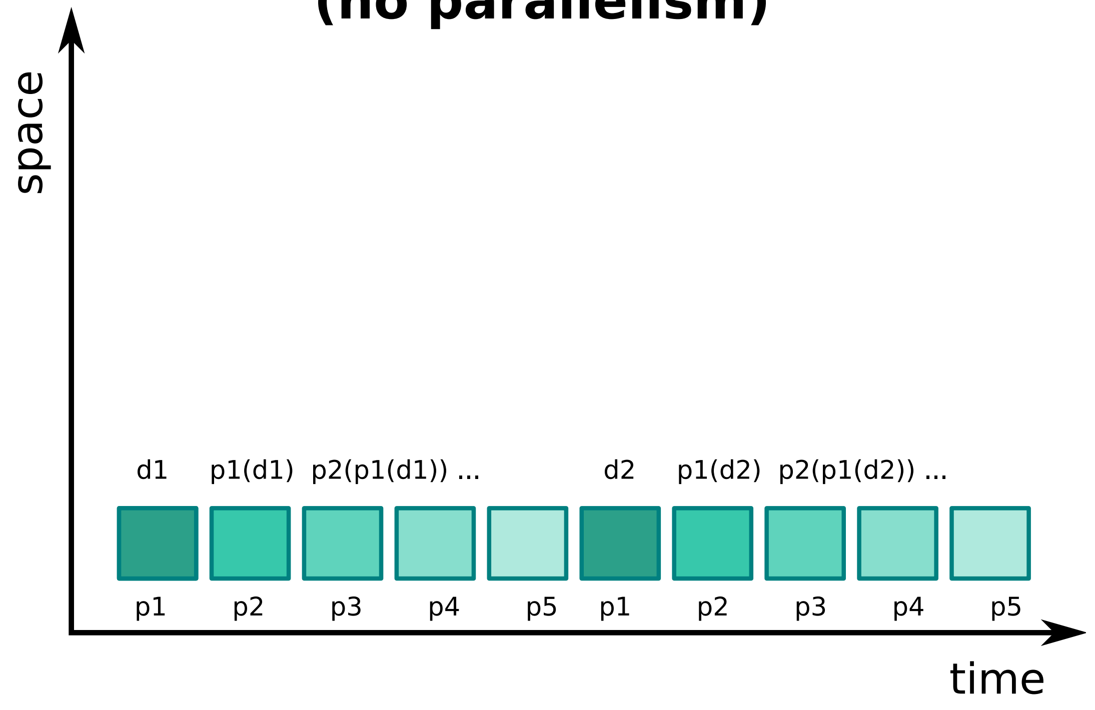
- Cores are not limited to microprocessor cores:
  - Graphics processors
  - DSP (crypto, codecs)
  - FPGA
- Design reuse is essential $\Rightarrow$ IP ("Intellectual Property") cores
  - IP cores are highly complex, self-contained units

Challenges for Multicore SoC programming

- Language and compiler developers have for years focussed on von Neumann machines (sequential memory-based processor)
- We need languages and compilers for parallel hardware
  - Support for parallelism
  - Separation of data flow from control flow
- The hardware should actively support the programming model

Processing modes

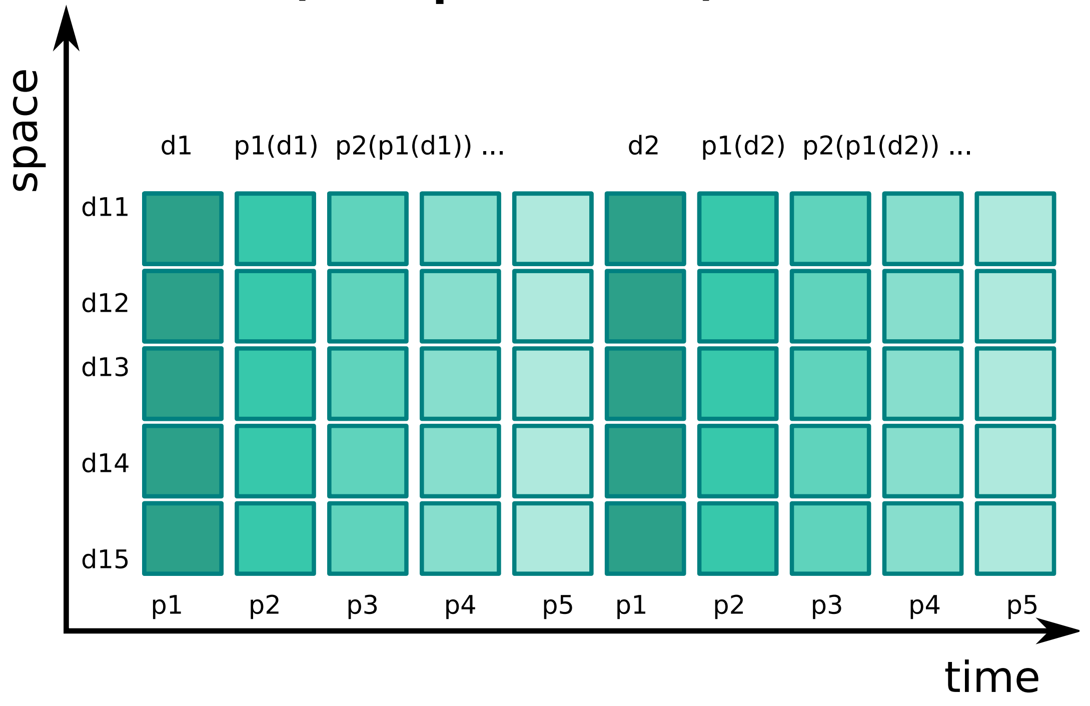
Serial processing (no parallelism)



$$\tau = \sum_{i=1}^{N} \tau_i \qquad (N\, processes)$$

Processing modes

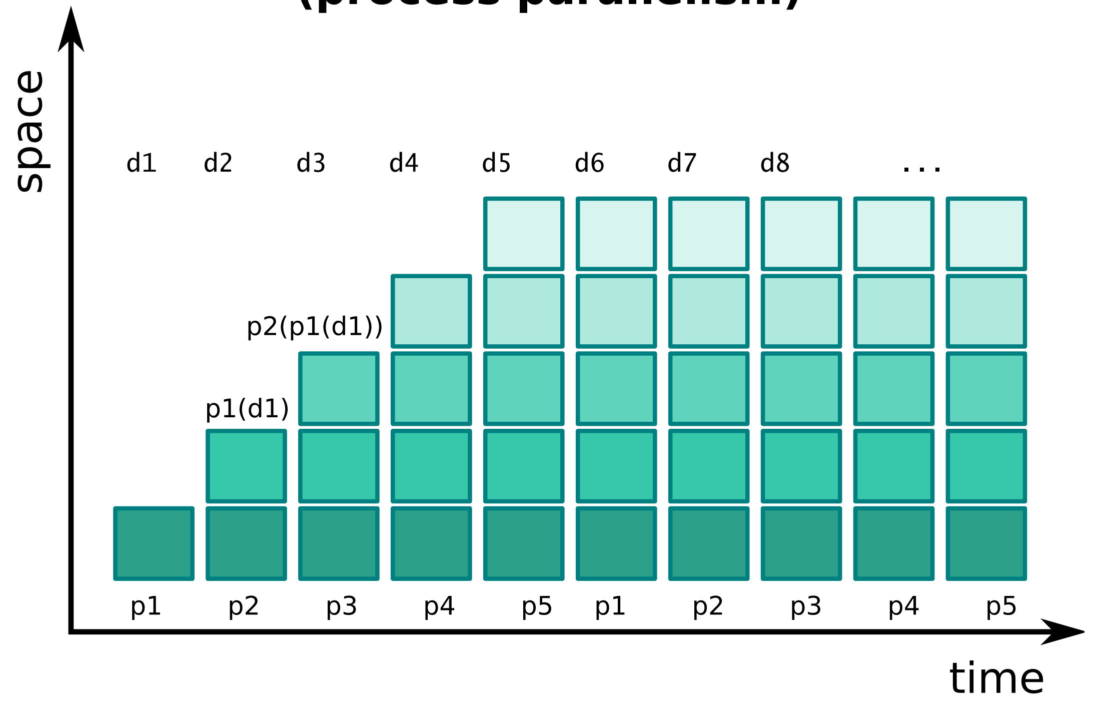
Parallel processing
(data parallelism)



$$\tau = \frac{1}{W}\sum_{i=1}^{N}\tau_i \qquad (N\, processes,\, width\, W)$$

Processing modes

Pipelined processing
(process parallelism)

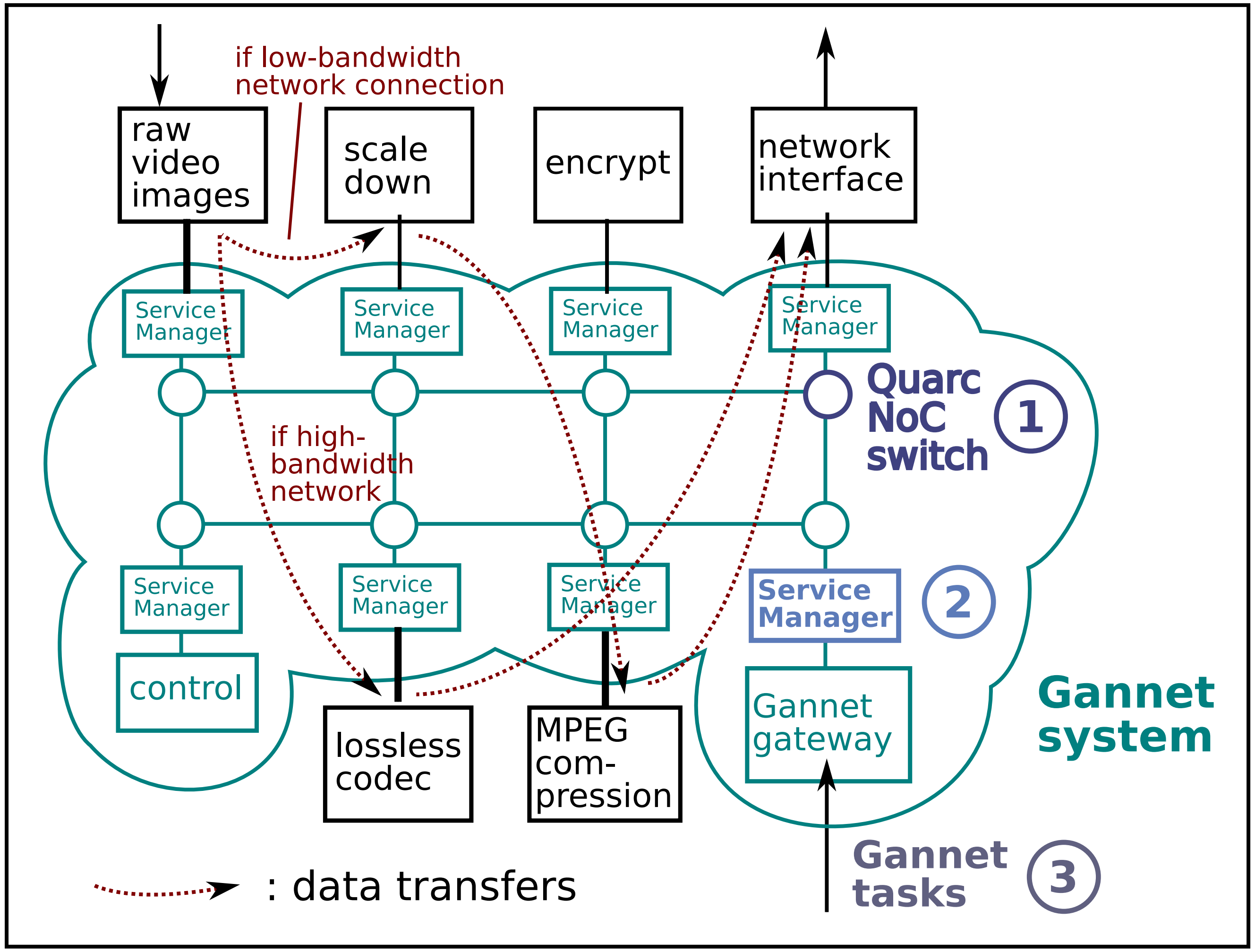


$$\tau = max(\tau_1, ..., \tau_N) \qquad (N\ processes)$$

- Heterogeneous Multicore SoC
- Gannet System-on-Chip
- Gannet language
- An operational semantics for the Gannet machine
- Streaming data processing

Gannet System-on-Chip

A simple video processing example:



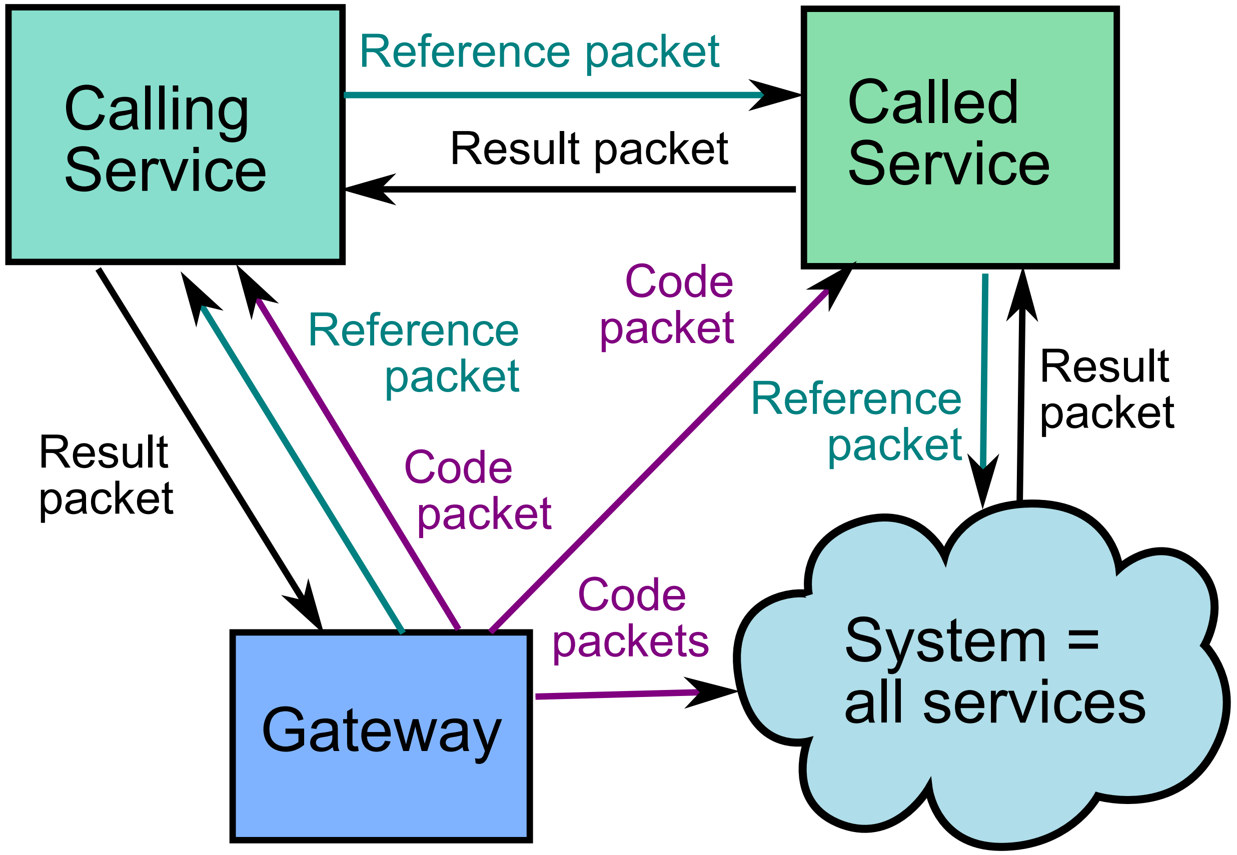
Gannet system architecture

- A **service-based** architecture for **heterogeneous Multicore SoCs:**
  - a collection of IP cores (HW/SW).
  - each IP core offers a a specific **service.**
  - IP cores acquire service behaviour through a generic data marshalling interface, the **Service Manager**
  - services interact through a Network-on-Chip (NoC)
- High abstraction-level design: high-level program governs behaviour of complete system

Gannet system operation

- The Gannet machine is a distributed computing system where every node (**service**) **consumes packets** and **produces packets** and can store state information between transactions.
- We denote a Gannet packet as $p(Type, To, Ret, Id; Payload)$
  - packet $Types$ are $code$, ref or $data$
- The operation of a Gannet **service** can be described in terms of
  - the **task code**
  - the internal state
  - the **result packet(s)** produced by the task

Gannet system operation



Gannet system operation

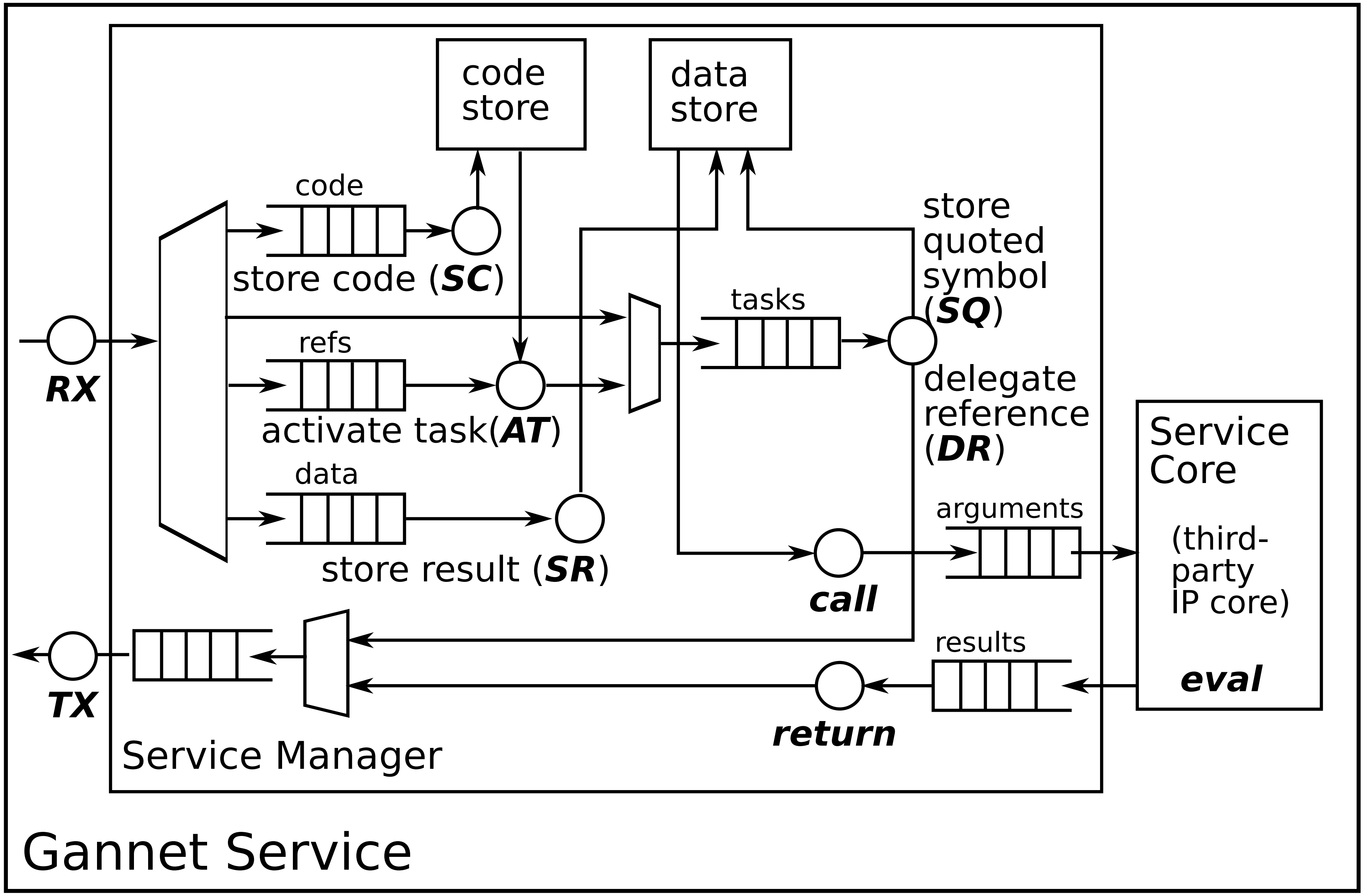
- SC: **Store code:** service S_i receives a **code** packet $p(code, S_i, S_j, R_{task}; t)$ where $t = (S_i\ a_1 ... a_n)$ and stores it referenced by R_{task}.
- AT: **Activate task :** the service S_i in $state_i$ receives a task **reference** packet $p(ref, S_i, S_j, R_{id}; R_{task})$

  the service activates the task referenced by R_{task}: $(S_i\ a_1 ... a_n)$. This results in evaluation of the arguments $a_1 .. a_n$:
  - DR: **Delegate reference:** the service manager delegates subtasks referenced by **reference sy**mbols via **reference** packets
  - SQ: **Store quoted symbol:** all **quoted** (i.e. constant) symbols in the code ares stored in the local store.
  - SR: **Store returned result:** result data from subtasks are stored in the local store.

Gannet system operation

- P: **Processing:** When all arguments of the subtask have been evaluated,
  - the data are passed on to the service core (**call**);
  - The core performs processing on the data (**eval**);
  - the service, now in $state'_i$, produces a result packet p_{res} (**return**) $p_{res} = p(Type_i, S_j, S_i, R_{id}; Payload_i)$ where both $Payload_i$ and the state change to $state'_i$ are the result of processing the evaluated arguments $a_1..a_n$ by the core of S_i.
- p_{res} is sent to S_j where $Payload_i$ is stored in a location referenced by R_{id}.

Gannet service architecture



- Heterogeneous Multicore SoC
- Gannet SoC
- Gannet language
- An operational semantics for the Gannet machine
- Streaming data processing

Gannet language

- The “assembly” language to program the Gannet system
- Intended as compilation target, not HLL
- No syntactic sugar
- A functional language, every service is mapped to an opaque function
- Gannet is a distributed machine for running this language

Gannet language

- Gannet syntax: S-expressions (cf. Lisp, Scheme)
- Simple video processing example:

```
(network-if
    (if (low-bandwidth)
       '(mpeg-compress (scale-down (raw-video))
       '(lossless-codec (raw-video))
    )
)
```

Control services in Gannet

- Any run-time reconfigurable system requires **control constructs** to be effective.
- In Gannet, these constructs (if/then, functions, blocks, variables, ...) are provided by services.
- Consequently, Gannet has no special forms apart from the quote.

Control services in Gannet

- Example with control services (factorial):

```
(let
    (assign 'fact
        (lambda 'n 'a 'f
          '(if (< n '2)
            'n
            '(apply f (- n '1) (* a n) 'f)
         )))
    (apply fact '4 '1 'fact)
)
```

Gannet-C

- **Gannet-C** is a HLL, but essentially a layer of syntactic sugar on top of Gannet, with C syntax to appeal to the masses...

```
{
    int fact(int,int);
    fact(n,a) = if(n<2) {
        n;
    } else {
        f(n-1,n*a);
    }
    fact(4,1);
}
```

Gannet language properties

- Some key properties of the Gannet language:
  - the evaluation order is unspecified
  - eager by default but lazy evaluation is possible
  - no side effects across services
- These properties
  - make the language fully concurrent (maximise parallelism)
  - and enable separation of control flow from data flow
  - allow efficient streaming data processing (pipelining)

Gannet language properties

- **Unspecified execution order:**
  - In a given function call it is not possible to predict the evaluation order of the arguments.
  - In practice, all arguments are evaluated in parallel; the call blocks until all arguments are ready.

```
(let
    (assign 'a (S1 ...))
    (assign 'b (S2 ...))
    (S3 ... b ...
        (S4 ... a ...)... )
)
```

Gannet language properties

- **Quoting and lazy evaluation:**
  - By default, Gannet is **eager**, i.e. it always evaluates all arguments before passing them on to the service core.
  - It should be possible to evaluate arguments at need (“**lazy**”).
  - Laziness is expressed by prefixing an expression or symbol with a single quote:

    ```
    (assign 'a (S1 ...))
    ```
  - Quoting causes the evaluation of the symbol to be **deferred** to the service core.

Gannet language properties

- **No side effects across services:**
  - A Gannet service is in general not a pure function, so calling a service can result in side effects.
  - In Gannet, side effects should be limited to changing the state of the service.
  - A call to a given service should not result in a modification of the state of the rest of the system.

- Heterogeneous Multicore SoC
- Gannet SoC
- Gannet language
- An operational semantics for the Gannet machine
- Streaming data processing

Gannet machine semantics

Notation

- $\bullet$: separates a packet from the other packets in the queue: $(p \bullet ps)$: packet at the head, $(ps \bullet p)$: packet at the tail.
- $*$ ("don't care"): the value of a field is not relevant.
- ... : presence of some non-specified entities (esp. in a store).
- _ : allocated space in a store

Gannet machine semantics

Definitions

- The Gannet system: N service nodes $S_i(...)$, $i \in 1..N$ and a gateway node $G(...)$.
- The unit of data transfer is the packet:

  $p = packet(Type, To, Ret, Id; Payload)$.

  $Payload$: **data** or an **expression** (depending on $Type$)

  Id : a Gannet symbol used as identifier for the payload

  To, Ret : addresses of destination and return services

Gannet machine semantics

- Packet receive and transmit FIFO queues: q_{RX} and q_{TX}

  RX queue: 4 queues muxed by packet $Type$:

  $q_{RX}(ps \bullet p)$ ::= $q_{RX}(...pt(ps \bullet p)...)$, $pt \in \{tasks, code, refs, data\}$
- Service node S_i also comprises:
  - The data store: $store_d(...(Id\ data)...)$
  - The task packet ($code$) store: $store_c(...(Id\ p)...)$

    p is the stored packet
  - The processing core $core(...)$

Gannet machine semantics

Thus an explicit notation for a service node S_i is:

$$S_i(q_{RX}(data(...), tasks(), code(...), refs(...)),\\ q_{TX}(...), store_d(...), store_c(...), core(...))$$

- All service nodes S_i operate **concurrently**
- The behaviour of a service S_i consists of a set of **actions**
- There is no central clock, all data transfers are asynchronous
- All actions are packet-driven

Packet transfer between services

The set of actions to transfer packets consists of:

- Transmit a packet (S_i): TX
- Receive a packet (S_j): RX

The semantics are straightforward:

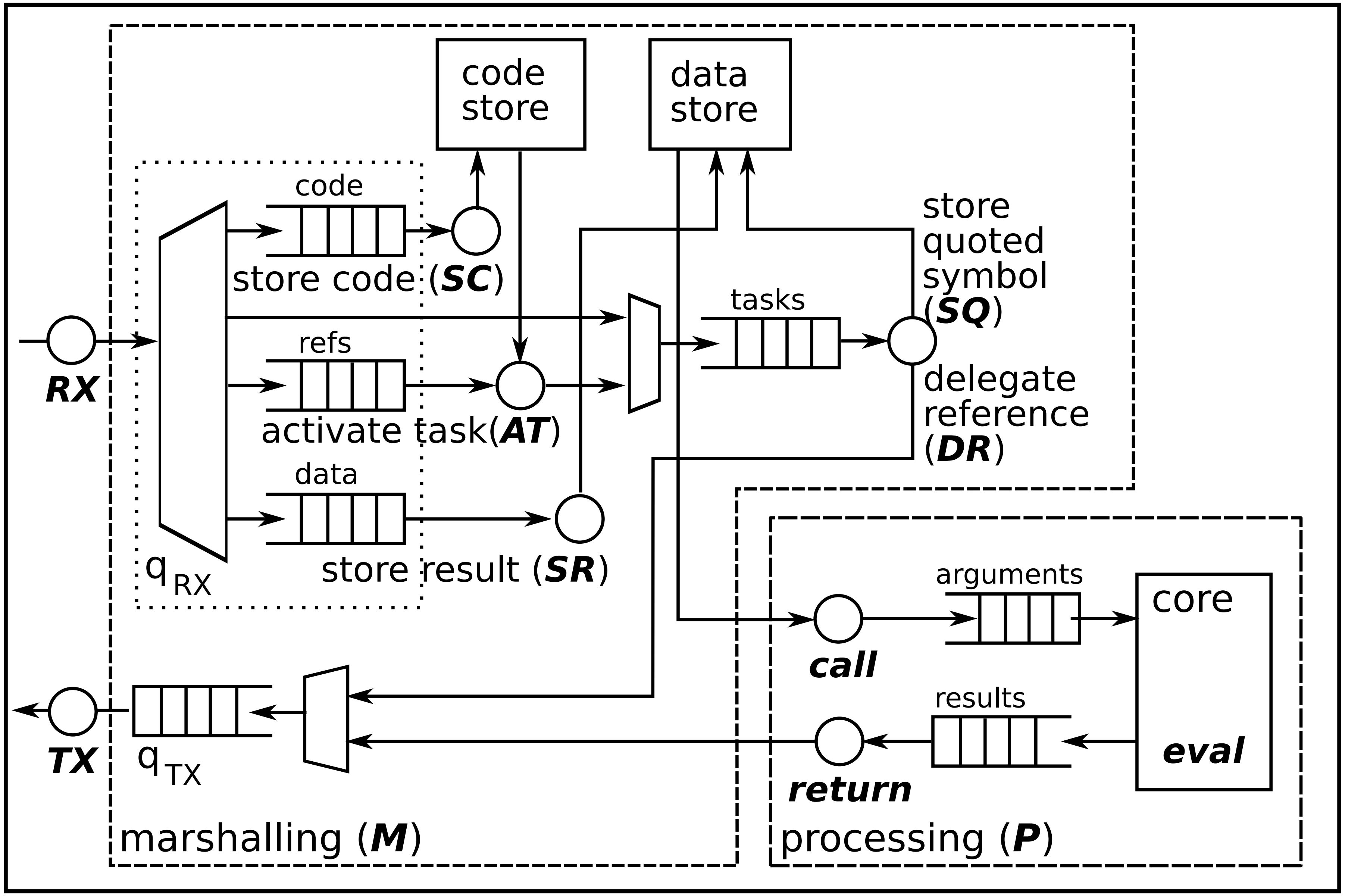
$$\begin{aligned} p &= packet(*, i, j, *; *) \\ S_i(q_{TX}(p \bullet ps)) &\longrightarrow^{TX} S_i(q_{TX}(ps)) \\ S_j(q_{RX}(qs)) &\longrightarrow^{RX} S_j(q_{RX}(qs \bullet p)) \end{aligned}$$

From now on, TX and RX are implicitly assumed.

The Gannet service manager

- On **receipt** of a packet, a service performs a set of **actions** which can result in
  - packets being **transmitted**
  - a change in the state of the store
- A **subset** of actions is performed by the **service manager**
  - The service manager is a **data marshalling** unit and **interface** between the service core and the system.
  - The service manager is **generic**: functionality is independent of the functionality of the service core.

Gannet service revisited



Service manager actions

Store code packet: SC

On receipt of a task, the gateway distributes code packets to the corresponding service nodes.

$$\begin{aligned} & p_{c,ij} = packet(code, i, G, r_{ij}; e_{ij});\ e_{ij} = \langle s_{ij} \ldots \rangle \\ & S_i(q_{RX}(code(p_{c,ij} \bullet ps_c)), store_{tp}(\ldots)) \\ \longrightarrow^{SC}\ & S_i(q_{RX}(code(ps_c)), store_{tp}(\ldots(r_{ij}\, p_{c,ij})\ldots)) \end{aligned}$$

Service manager actions

Activate task: AT

$$p_{r,i} = packet(reference, i, j, r_i'; r_i)$$

$$p_{c,i} = packet(code, i, *, r_i; se_i);\; se_i = \langle s_i \ldots r_j \ldots \rangle$$

$$S_i(q_{RX}(tasks(qs), refs(p_{r,i} \bullet ps)), store_c(\ldots(r_i\, p_{c,i})\ldots))$$

$$\longrightarrow^{AT} S_i(q_{RX}(tasks(qs \bullet p_{t,i}), refs(ps)), store_c(\ldots))$$

$$p_{t,i} = packet(task, i, j, r_i'; se_i)$$

Marshalling action set M

Delegate reference: DR

$$p_i = packet(task, i, *, *; se_i);\ se_i = \langle s_i \ldots r_j \ldots \rangle$$

$$S_i(q_{RX}(tasks(p_i \bullet ps)), q_{TX}(qs), store(\ldots))$$

$$\xrightarrow{}^{DR}\ S_i(q_{RX}(tasks(ps)), q_{TX}(qs \bullet p_{r,j}), store(\ldots(r_j\_)\ldots))$$

$$p_{r,j} = packet(reference, j, i, r_j; r_j)$$

Marshalling action set M

Store returned result: SR

$$p_i = packet(data, i, *, s_j; w_j)$$

$$S_i(q_{RX}(data(p_i \bullet ps)), store(\ldots(s_j \_)\ldots))$$

$$\xrightarrow{}^{SR} S_i(q_{RX}(data(ps)), store(\ldots(s_j w_j)\ldots))$$

Marshalling action set M

Store quoted symbol: SQ

$$p_i = packet(task, i, *, *; se_i);\ se_i = \langle s_i ... qr_j ... \rangle$$

$$S_i(q_{RX}(tasks(p_i \bullet ps)), store(...))$$

$$\xrightarrow{}^{SQ} S_i(q_{RX}(tasks(ps)), store(...(qr_j\, r_j)...))$$

Marshalling action set M

The action of the complete M set $\{SC, AT, DR, SR, SQ\}$ can be abstracted as:

$$p_i = packet(task, i, *, *; se_i);\ se_i = \langle s_i\ a_1 ... a_n \rangle$$

$$a_i ::= qr_i \mid r_i$$

$$S_i(q_{RX}(p_i \bullet ps), q_{TX}(qs), store(...))$$

$$\xrightarrow{M}\ S_i(q_{RX}(ps), q_{TX}(qs), store(...(a_1\, wr_1)...(a_n\, wr_n)...))$$

$$wr_i ::= w_i \mid r_i$$

Processing action set P

- The actions of the **service core** determine the **functionality** of the service.
- This functionality can be defined as the type, destination and payload content of the packet produced and the state change of the store based on the values marshalled by the service manager.
- For data-processing services, the service core implements a function cs_i which takes n arguments with values $w_1 ... w_n$ and produces a result w.

Processing action set P

The P set consists of the actions $\{call, eval, return\}$:

$$
\begin{aligned}
& S_i(q_{TX}(ps), store((s_1\, w_1)\ldots(s_n\, w_n)\ state), core()) \\
\longrightarrow^{call}\ & S_i(q_{TX}q(ps), store(state), core(\,(cs_i\, w_1 \ldots w_n)\,)) \\
\longrightarrow^{eval}\ & S_i(q_{TX}(ps), store(state'), core(\,w\,)) \\
\longrightarrow^{return}\ & S_i(q_{TX}(ps \bullet p), store(state'), core()) \\
& p = packet(*,*,i,*;w)
\end{aligned}
$$

Processing action set P

The P set actions can be abstracted as:

$$
\begin{aligned}
& S_i(q_{RX}(qs), q_{TX}(ps), store((s_1\, w_1)\ldots(s_n\, w_n)\ state)) \\
\longrightarrow^{P}\ & S_i(q_{RX}(qs), q_{TX}(ps \bullet p), store(state')) \\
& p = packet(*, *, i, *; w);\ (cs_i\, w_1 \ldots w_n) \rightarrow w
\end{aligned}
$$

Service semantics

- All actions can be abstracted to the M and P sets
- The semantics of a service can be described completely in terms of
  - the task and results packets
  - the state of the store

Non-control service semantics

- **Non-control services** are services of which the core behaviour can be modelled as delta application
  - the resulting packet will be of type **data**
  - the state of the **store** is not modified by the evaluation

$$p_{rx} = packet(task, i, j, r_j; e_i);\ e_i = \langle s_i \ldots r_j \ldots \rangle;\ r_j \rightarrow w_j$$

$$S_i(store(\ldots))$$

$$\xrightarrow{M}\ S_i(store(\ldots(r_j w_j)\ldots))$$

$$\xrightarrow{P}\ S_i(store(\ldots))$$

$$p_{tx} = packet(data, j, i, r_j; w_i);\ w_i = \delta(s_i, \ldots, w_j, \ldots)$$

Control service semantics

- **Control services** provide control/language constructs to the Gannet system
- Evaluation of a task by a control service can result in
  - the creation of a result packet of type **task, code** or **reference**
  - a change of the **state** of the **store**
- Example: streaming data processing

- Heterogeneous Multicore SoC
- Task control in NoC-based SoCs
- Gannet SoC
- Gannet language
- An operational semantics for Gannet
- Streaming data processing

Streaming data processing

- Streaming data processsing is very important in SoC applications: audio processing, video processing, network traffic processing
- Gannet provides streaming data processing with run-time reconfigurable data paths
- As the system is distributed, there is no shared-memory bottleneck

Pipelining

- Efficient streaming data processing requires a pipeline
- Pipelines are constructed by inserting intermediate buffers between the processes
- Assume N processes, each process i takes a time τ_i
  - Throughput without pipeline: $1/\sum_{i=1}^{N} \tau_i$
  - Throughput with pipeline: $1/max(\tau_1, ..., \tau_N)$
- If τ is the same for every stage, the difference in throughput is a factor N

Data streams in Gannet

Gannet-C

The `Stream` type indicates that every call of this service can return a different value. Results of nested function calls on a `Stream` type are automatically pipelined.

```
Stream<int> stream_in();
void stream_out(float);
float s1(int);
int s2(int); int s3(int);
while () {
    stream_out(s1(s2(s3(stream_in()))));
}
```

Data streams in Gannet

Gannet

`(buf 'b (S...))` is syntactic sugar for `(S-buf 'b ...);`

`(stream 'b)` is syntactic sugar for `(S-stream 'b)`

```
(let
    (buf 'b3 (s3 (stream_in)))
    (buf 'b2 (s2 (stream 'b3)))
    (buf 'b1 (s1 (stream 'b2)))
    (label L '(return 'L
        (stream_out (stream 'b1))
    ))
)
```

Data streams in Gannet

Buffering semantics

The Gannet machine implements streaming by means of buffers in every service:

$$p_{buf} = packet(task, S_3, let, r; e_{buf});\ e_{buf} = \langle \mathbf{S_3^{buf}}\, qb \ldots e_{stream} \ldots \rangle;$$

$$qb \rightarrow b; e_{stream} \rightarrow w_{stream}$$

$$S(q_{RX}(p_{buf} \bullet ps), q_{TX}(qs), store(\ldots))$$

$$\longrightarrow^{M}\ S(store(\ldots(r_{stream}\, w_{stream})\,(qb\ b)\ldots))$$

$$\longrightarrow^{P}\ S(store(\ldots(b\,(w, p_{buf}))\ldots)); w = \delta(S, \ldots, w_{stream}, \ldots)$$

$$p_r = packet(data, let, S_3, r; qb)$$

Data streams in Gannet

Streaming semantics

The `stream` call returns the buffered result and reschedules the buffering task:

$$p_{buf} = packet(task, S_3, let, r; e_{buf});\ e_{buf} = \langle \mathbf{S_3^{buf}}\, qb \ldots e_{stream} \ldots \rangle;$$

$$qb \rightarrow b; e_{stream} \rightarrow w_{stream}$$

$$p_{stream} = packet(task, S_3, S_2, r_2; e_{stream});\ e_{stream} = \langle \mathbf{S_3^{stream}}\, qb \rangle;$$

$$S(q_{RX}(p_{stream} \bullet ps), q_{TX}(qs), store(\ldots(b\,(w, p_{buf}))\ldots))$$

$$\xrightarrow{}^{M} S(store(\ldots(qb\, b)\ldots(b\,(w, p_{buf}))\ldots))$$

$$\xrightarrow{}^{P} S(q_{RX}(ps \bullet p_{buf}), q_{TX}(p_w), store(\ldots));$$

$$p_w = packet(data, S_2, S_3, r_2; w)$$

More on buffers and streams

- The buffer/stream mechanism allows interleaving (multiplexing) of streams on every service in the pipeline
- Additional buffer control via `get`, `peek` and end-of-stream detection
- The buffer mechanism also serves to implement caching (call-by-need) and accumulation

Conclusion

- Heterogeneous multicore SoCs are the next step in evolution of processors
- Gannet proposes an architecture, language and programming model for such systems
- The Gannet system supports streaming data processing, data parallelism and distributed flow control and can be programmed using a high-level language, Gannet-C.
- We have presented an operational semantics to formally describe the operation of the Gannet machine.
- As an illustration, we have presented the semantics for streaming data processing.