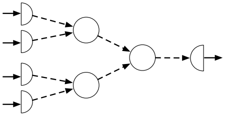
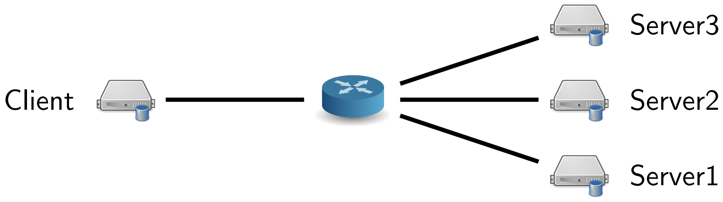
FLICK: Developing and Running Application-Specific Network Services

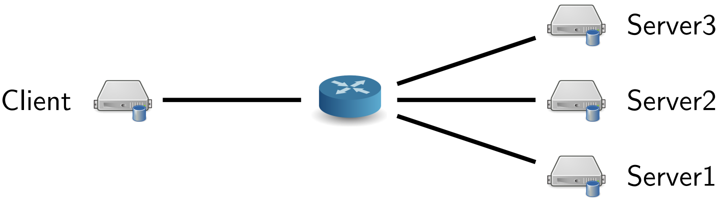


Presenter: Richard G. Clegg, Imperial College
Imperial College: Abdul Alim, Luo Mai, Lukas Rupprecht, Eric Seckler, Paolo Costa, Peter Pietzuch, Alexander L. Wolf
Cambridge: Nik Sultana, Jon Crowcroft, Anil Madhavapeddy, Andrew W. Moore, Richard Mortier
Nottingham: Masoud Koleini, Carlos Oviedo, Derek McAuley
Kent: Matteo Migliavacca

Packet processing vs application-specific middlebox



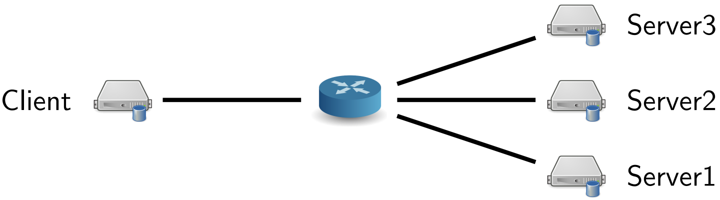
Packet processing vs application-specific middlebox



Packet processing
(ECMP loadbalancer)

```
process(packet):
  dest=hash(packet.srcIP + packet
      .srcport)
  forward(packet,dest);
```

Packet processing vs application-specific middlebox

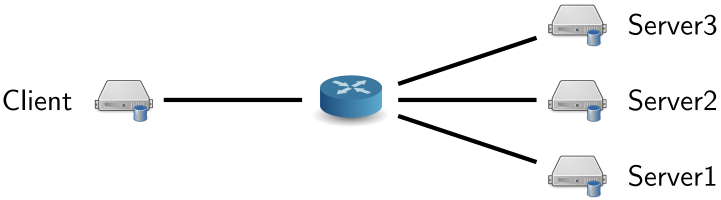


Packet processing
(ECMP loadbalancer)

```
process(packet):
  dest=hash(packet.srcIP + packet
      .srcport)
  forward(packet,dest);
```

- Header data only used.
- Packets have fixed format.
- Basic data unit is packet.

Packet processing vs application-specific middlebox



Packet processing
(ECMP loadbalancer)

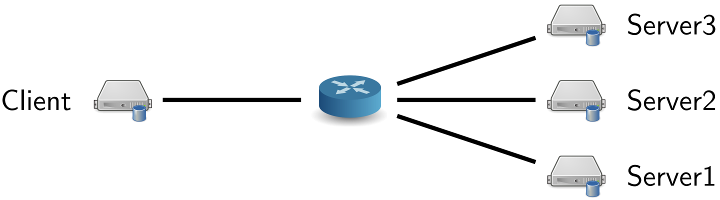
```
process(packet):
  dest=hash(packet.srcIP + packet
      .srcport)
  forward(packet,dest);
```

Application-specific
(memcached router)

```
process(key_val_pair):
  dest=hash(key_val_pair.key);
  forward(key_val_pair,dest);
```

- Header data only used.
- Packets have fixed format.
- Basic data unit is packet.

Packet processing vs application-specific middlebox



Packet processing
(ECMP loadbalancer)

```
process(packet):
  dest=hash(packet.srcIP + packet
      .srcport)
  forward(packet,dest);
```

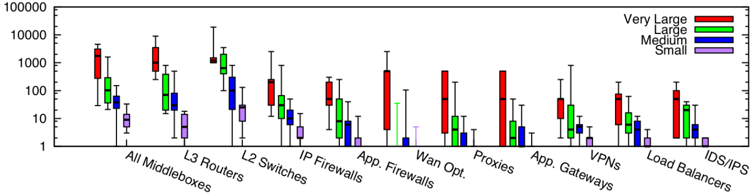
- Header data only used.
- Packets have fixed format.
- Basic data unit is packet.

Application-specific
(memcached router)

```
process(key_val_pair):
  dest=hash(key_val_pair.key);
  forward(key_val_pair,dest);
```

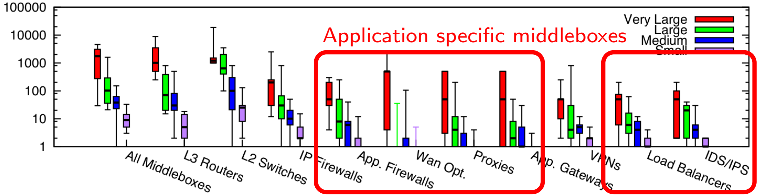
- Applications have different data formats (e.g. key-value pairs, HTTP request/reply).
- TCP flow not packets.
- One packet != one data item.

Problem: The application-specific middlebox



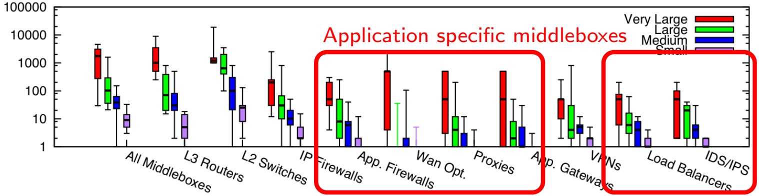
Figures from: Making Middleboxes Someone Elses Problem, Sherry et al. SIGCOMM 2012

Problem: The application-specific middlebox



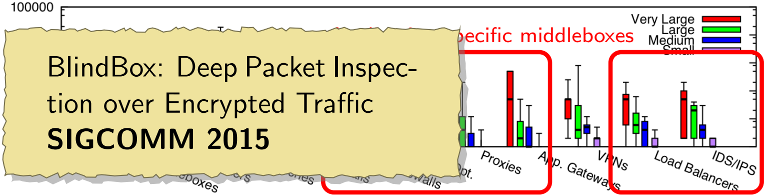
Figures from: Making Middleboxes Someone Elses Problem, Sherry et al. SIGCOMM 2012

Problem: The application-specific middlebox



NetAgg: Using Middleboxes for On-path Aggregation
CoNEXT 2014

Figures from: Making Middleboxes Someone Elses Problem, Sherry et al. SIGCOMM 2012

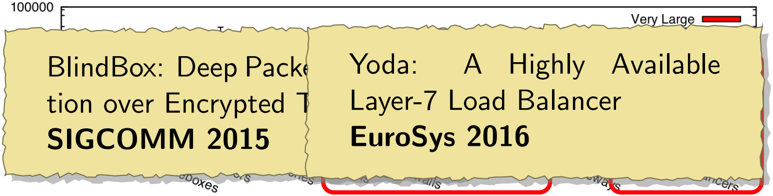


BlindBox: Deep Packet Inspection over Encrypted Traffic
SIGCOMM 2015

NetAgg: Using Middleboxes for On-path Aggregation
CoNEXT 2014

Figures from: Making Middleboxes Someone Elses Problem, Sherry et al. SIGCOMM 2012

Problem: The application-specific middlebox



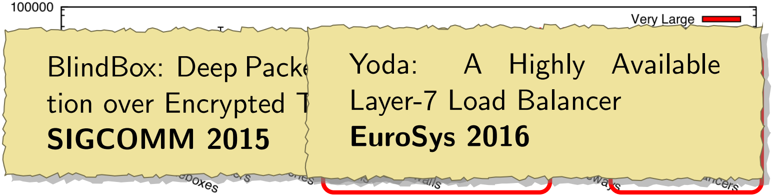
BlindBox: Deep Packe
tion over Encrypted T
SIGCOMM 2015

Yoda: A Highly Available Layer-7 Load Balancer
EuroSys 2016

NetAgg: Using Middleboxes for On-path Aggregation
CoNEXT 2014

Figures from: Making Middleboxes Someone Elses Problem, Sherry et al. SIGCOMM 2012

Problem: The application-specific middlebox



BlindBox: Deep Packe
tion over Encrypted T
SIGCOMM 2015

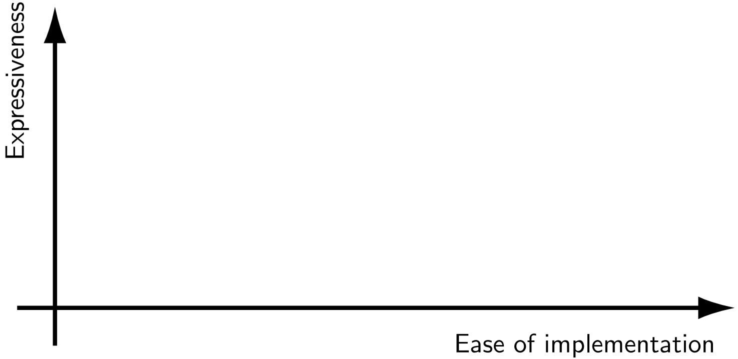
Yoda: A Highly Available Layer-7 Load Balancer
EuroSys 2016

NetAgg: Using Mid
for On-path Aggregat
CoNEXT 2014

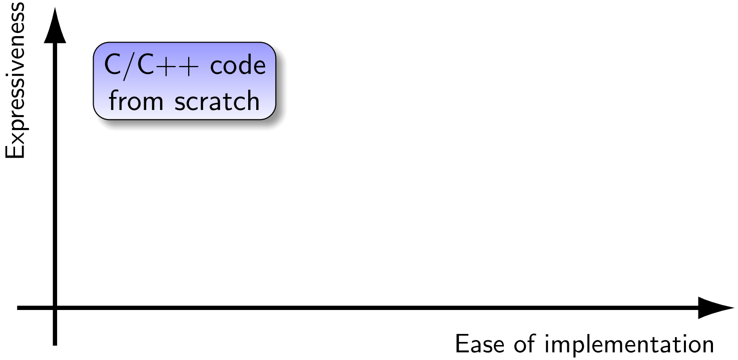
Introducing mcrouter:
A memcached protocol router
Facebook blog

Figures from: Making Middleboxes Someone Elses Problem, Sherry et al. SIGCOMM 2012

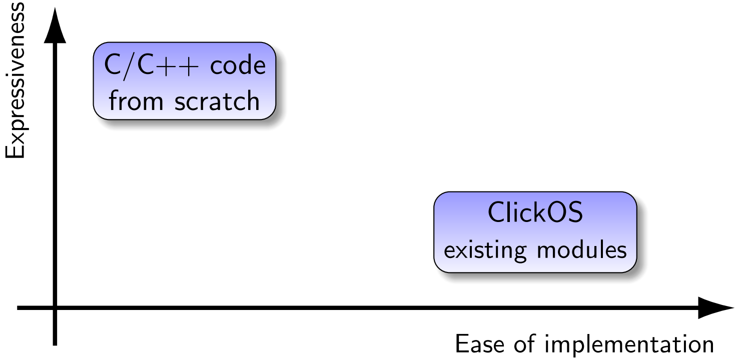
Creating new application-specific middlebox



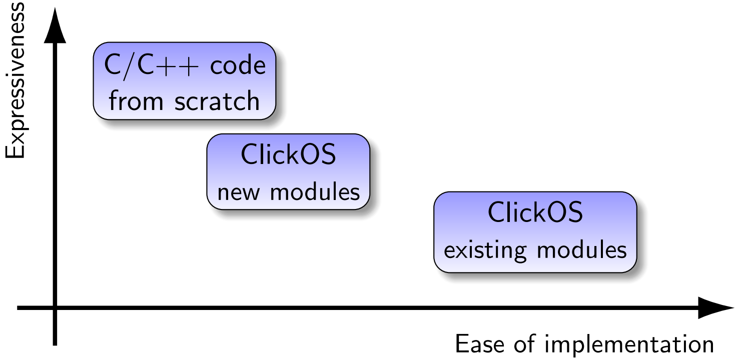
Creating new application-specific middlebox



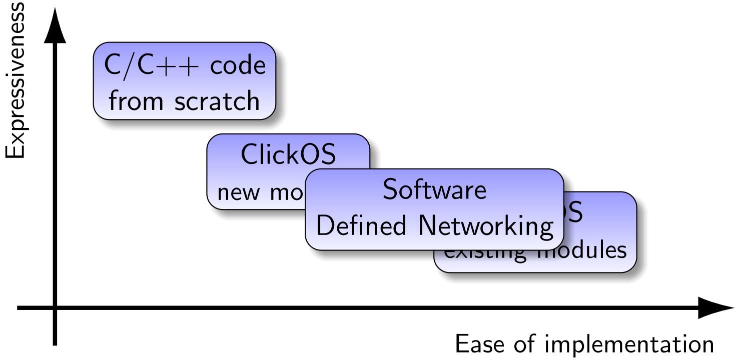
Creating new application-specific middlebox



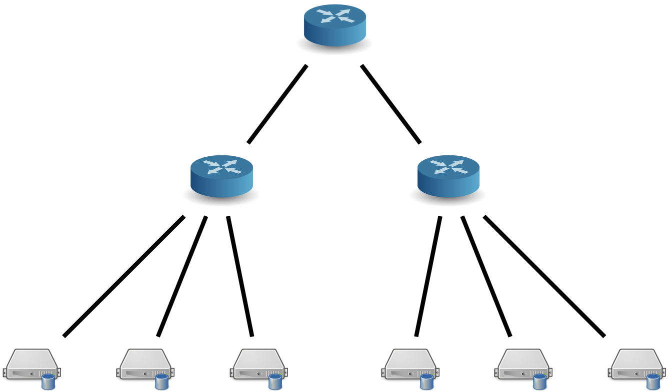
Creating new application-specific middlebox



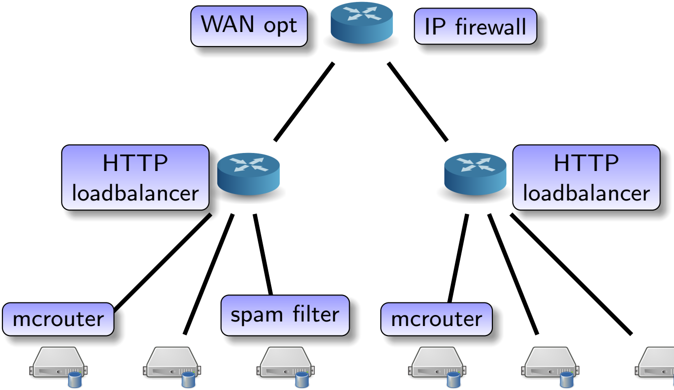
Creating new application-specific middlebox



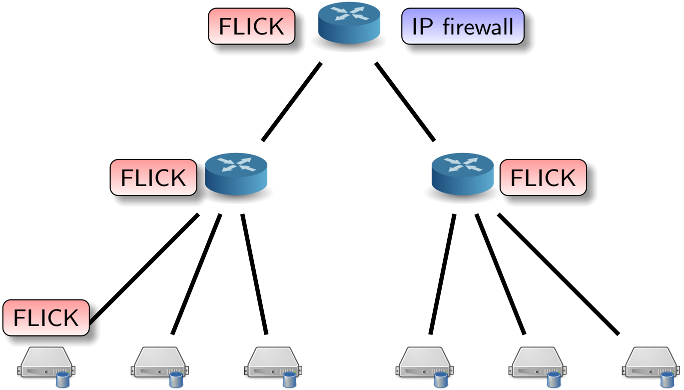
FLICK for the datacentre



FLICK for the datacentre



FLICK for the datacentre



General system for application-specific middleboxes?

Challenge 1: Ease-of-use

Rapidly express many middlebox functions.
System created in hours not weeks/months.

General system for application-specific middleboxes?

Challenge 1: Ease-of-use

Rapidly express many middlebox functions.
System created in hours not weeks/months.

Challenge 2: Performance

Generality must not have large performance penalty.
Performance similar to specially written system.

General system for application-specific middleboxes?

Challenge 1: Ease-of-use

Rapidly express many middlebox functions.
System created in hours not weeks/months.

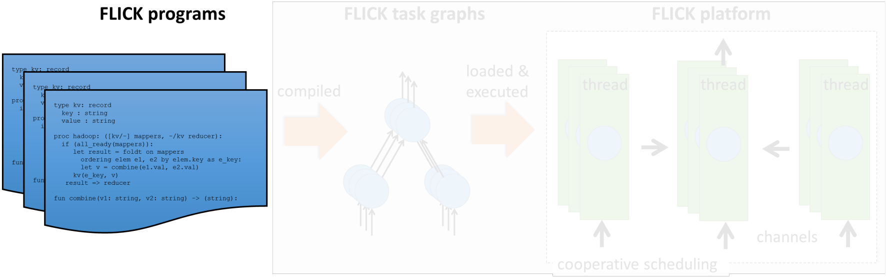
Challenge 2: Performance

Generality must not have large performance penalty.
Performance similar to specially written system.

Challenge 3: Safety/Isolation

Middleboxes should be "safe" in resource usage.
Applications on same machine share resources well.

FLICK overview

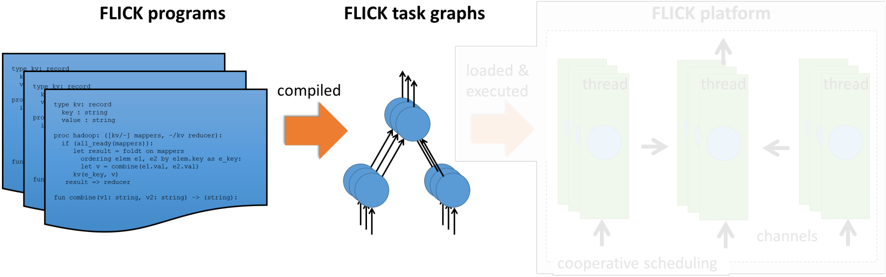


Flick programs

Domain specific language (DSL) for application-specific middleboxes.

Tens of lines of code not tens of thousands

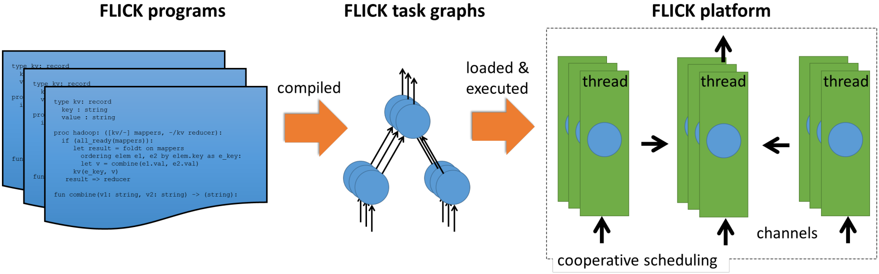
FLICK overview



Flick task graphs

Break work into independently schedulable units (tasks).
Join tasks by channels into task graphs.

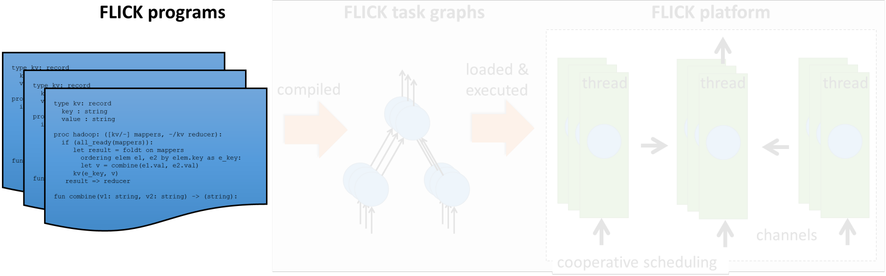
FLICK overview



Flick platform

The running implementation. Integrates the compiled C++ from DSL. Handles network connections, worker threads and scheduling tasks.

FLICK – the language



FLICK (language) – features

```
type cmd: record
 key    : string

proc Memcached: (cmd/cmd client, [cmd/cmd] backends)
  | backends => client
  | client => target_backend(backends)

fun target_backend: ([-/cmd] backends, req:cmd) -> ()
  let target = hash(req.key) mod len(backends)
 req => backends[target]
```

FLICK (language) – features

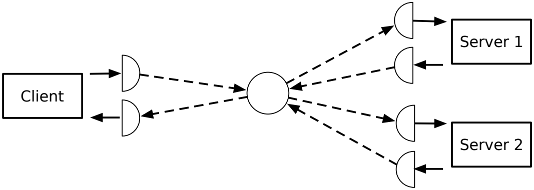
```
type cmd: record
 key    : string

proc Memcached: (cmd/cmd client, [cmd/cmd] backends)
  | backends => client
  | client => target_backend(backends)

fun target_backend: ([-/cmd] backends, req:cmd) -> ()
  let target = hash(req.key) mod len(backends)
 req => backends[target]
```

- Process as basic unit of code expresses flow of typed data.
- Control structures restricted. Bounded loops and hence execution time.
- Strongly typed for safety.

FLICK (language) – processing data (memcached)

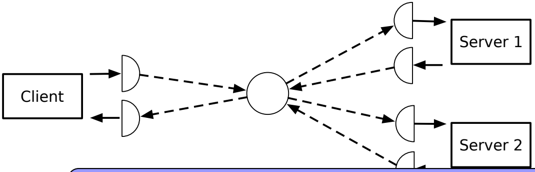



```
type cmd: record
 key    : string

proc Memcached: (cmd/cmd client, [cmd/cmd] backends)
  | backends => client
  | client => target_backend(backends)

fun target_backend: ([-/cmd] backends, req:cmd) -> ()
  let target = hash(req.key) mod len(backends)
 req => backends[target]
```

FLICK (language) – processing data (memcached)



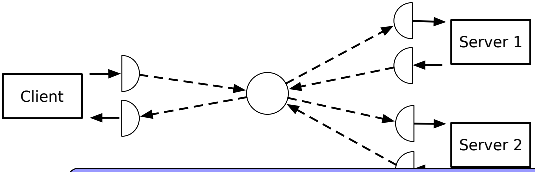
Structure allows work to
break into smaller task units

```
type
 key    : string

proc Memcached: (cmd/cmd client, [cmd/cmd] backends)
   | backends => client
   | client => target_backend(backends)

fun target_backend: ([-/cmd] backends, req:cmd) -> ()
  let target = hash(req.key) mod len(backends)
 req => backends[target]
```

FLICK (language) – processing data (memcached)



Structure allows work to
break into smaller task units

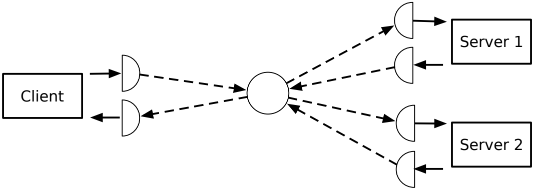
```
type
 key    : string

proc Memcached: (cmd/cmd client, [cmd/cmd] backends)
  | backends => client
  | c
```

Convenient abstractions for middlebox

```
fun target_backend: ([-/cmd] backends, req:cmd) -> ()
  let target = hash(req.key) mod len(backends)
 req => backends[target]
```

FLICK (language) – processing data (memcached)



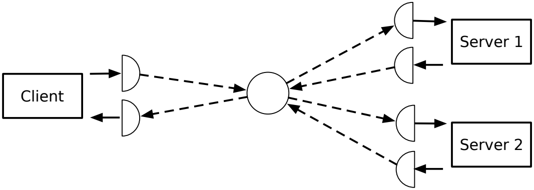
Type definition only necessary fields included

```
type cmd: record
 key    : string
```

```
proc Memcached: (cmd/cmd client, [cmd/cmd] backends)
  | backends => client
  | client => target_backend(backends)

fun target_backend: ([-/cmd] backends, req:cmd) -> ()
  let target = hash(req.key) mod len(backends)
 req => backends[target]
```

FLICK (language) – processing data (memcached)



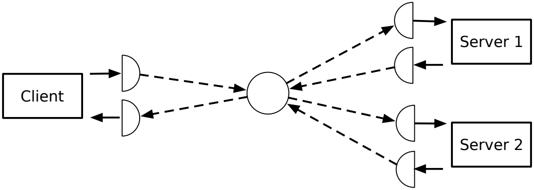
Process: entry point defines how channels connect

```
type cmd: record
 key    : string
```

```
proc Memcached: (cmd/cmd client, [cmd/cmd] backends)
  | backends => client
  | client => target_backend(backends)
```

```
fun target_backend: ([-/cmd] backends, req:cmd) -> ()
  let target = hash(req.key) mod len(backends)
 req => backends[target]
```

FLICK (language) – processing data (memcached)



Function selects backend using key

```
type cmd: record
 key    : string

proc Memcached: (cmd/cmd client, [cmd/cmd] backends)
  | backends => client
  | client => target_backend(backends)
```

```
fun target_backend: ([-/cmd] backends, req:cmd) -> ()
  let target = hash(req.key) mod len(backends)
 req => backends[target]
```

FLICK (language) – parsing data (memcached)

```
type cmd = unit {
    %byteorder = big;
    magic_code           : uint8;
    opcode               : uint8;
    key_len              : uint16;
    extras_len           : uint8;
                         : uint8;  # anon field - future use
    status_or_v_bucket : uint16;
    total_len            : uint32;
    opaque               : uint32;
    cas                  : uint64;
    extras    : bytes &length = self.extras_len;
    key       : string &length = self.key_len;
    value     : bytes &length = self.value_len;
};
```

FLICK (language) – parsing data (memcached)

```
type cmd = unit {
    %
    m
    opcode                : uint8;
    key_len               : uint16;
    extras_len            : uint8;
                          : uint8;  # anon field - future use
    status_or_v_bucket : uint16;
    total_len             : uint32;
    opaque                : uint32;
    cas                   : uint64;
    extras    : bytes &length = self.extras_len;
    key       : string &length = self.key_len;
    value     : bytes &length = self.value_len;
};
```

Based on Spicy/binpac++ [IMC2006]

FLICK (language) – parsing data (memcached)

```
type cmd = unit {
    %
    m
    opcode                  : uint8;
    key_len                 : uint16;
    extras_len              : uint8;
                                                          ure use
    s
    t
    opaque                  : uint32;
    cas                     : uint64;
    extras    : bytes &length = self.extras_len;
    key       : string &length = self.key_len;
    value     : bytes &length = self.value_len;
};
```

Based on Spicy/binpac++ [IMC2006]

Developer can quickly parse even complex formats like HTTP

FLICK (language) – parsing data (memcached)

```
type cmd = unit {
    %
    m
    opcode                : uint8;
    key_len               : uint16;
    extras_len            : uint8;
                                                   ure use
    s
    t
    opaque                : uint32;
    cas                   : uint64;
    extras    : bytes &length = self.extras_len;
    k
    v
};
```

Based on Spicy/binpac++ [IMC2006]

Developer can quickly parse even complex formats like HTTP

Compiles to efficient C++. Only extracts fields used in processing

FLICK (language) – parsing data (memcached)

```
type cmd = unit {
    %byteorder = big;
    magic_code          : uint8;
    opcode              : uint8;
    key_len             : uint16;
    extras_len          : uint8;
                        : uint8;  # anon field - future use
    status_or_v_bucket  : uint16;
    total_len           : uint32;
    opaque              : uint32;
    cas                 : uint64;
    extras    : bytes &length = self.extras_len;
    key       : string &length = self.key_len;
    value     : bytes &length = self.value_len;
};
```

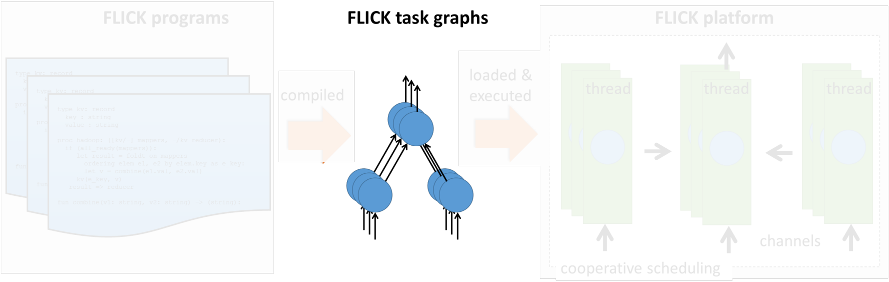
Fixed width field easy to define

FLICK (language) – parsing data (memcached)

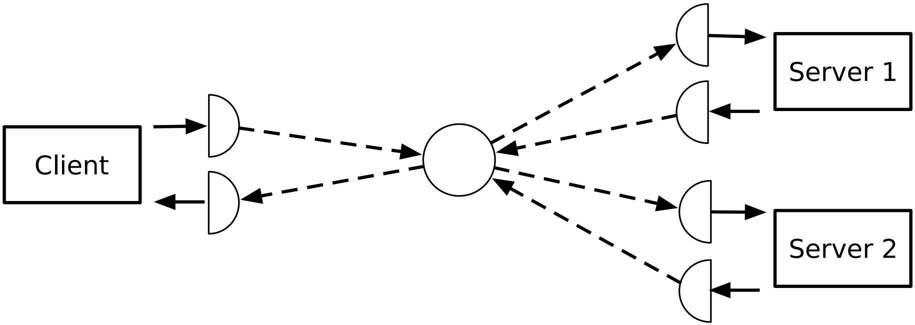
Field length depends on previous field

```
type cmd = unit {
    %byteorder = big;
    magic_code          : uint8;
    opcode              : uint8;
    key_len             : uint16;
    extras_len          : uint8;
                        : uint8;  # anon field - future use
    status_or_v_bucket  : uint16;
    total_len           : uint32;
    opaque              : uint32;
    cas                 : uint64;
    extras    : bytes &length = self.extras_len;
    key       : string &length = self.key_len;
    value     : bytes &length = self.value_len;
};
```

FLICK – the task graph

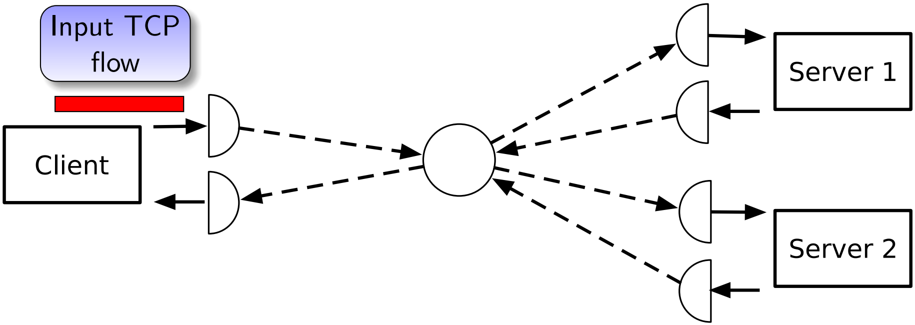


FLICK – the task graph

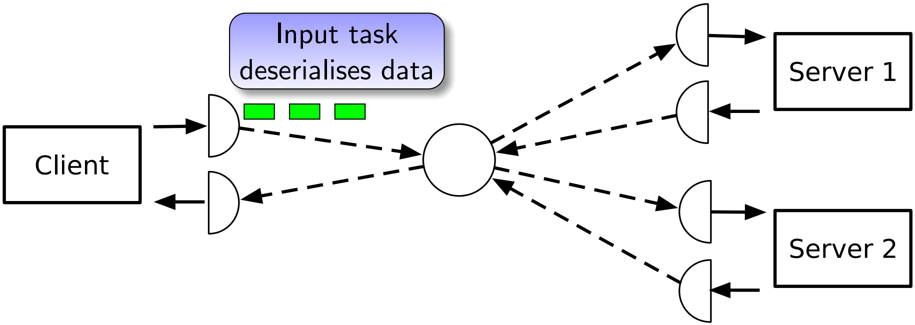


Separate input, processing and
output tasks enable parallelism

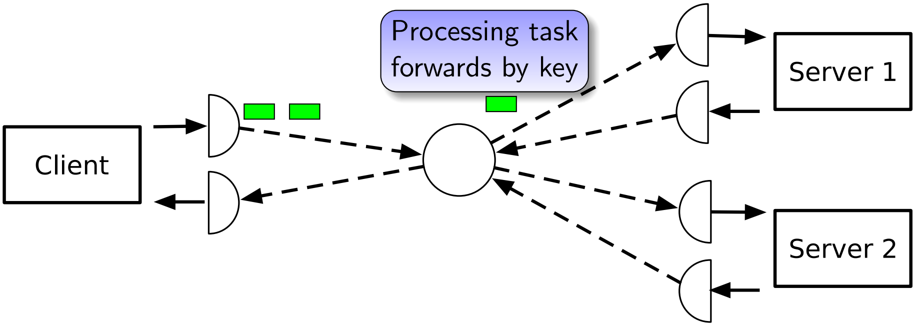
FLICK – the task graph



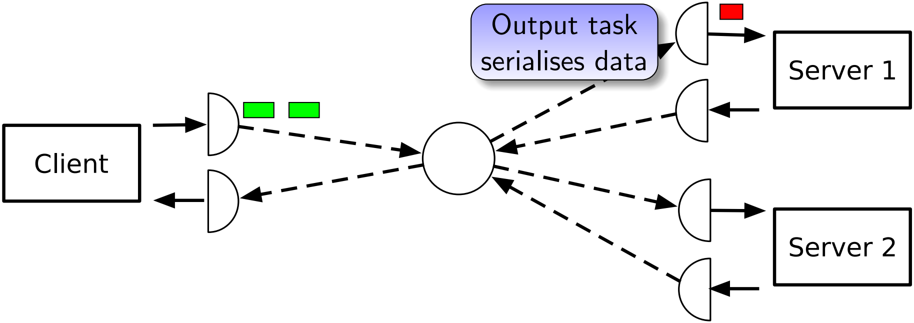
FLICK – the task graph



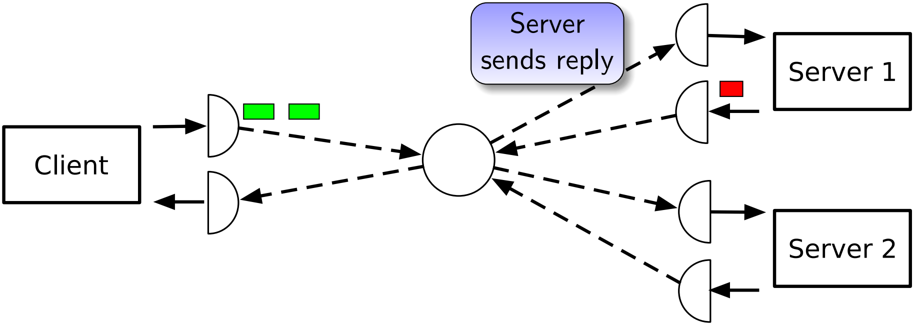
FLICK – the task graph



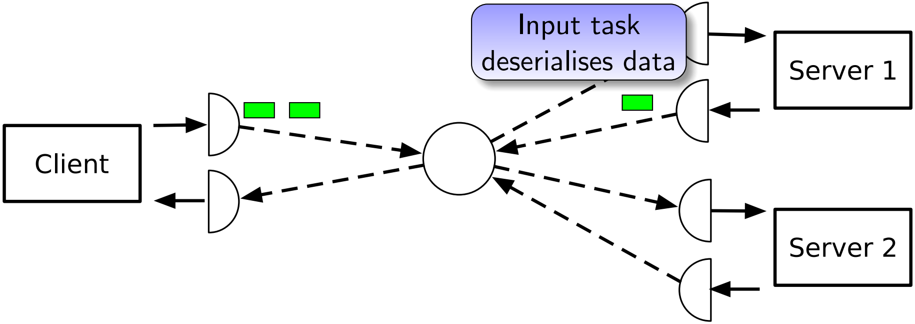
FLICK – the task graph



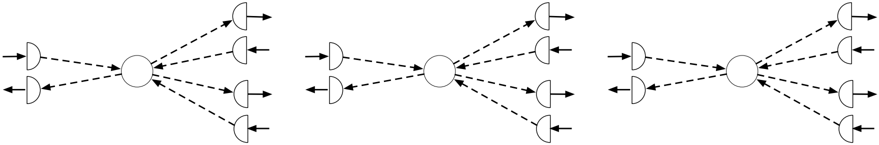
FLICK – the task graph



FLICK – the task graph

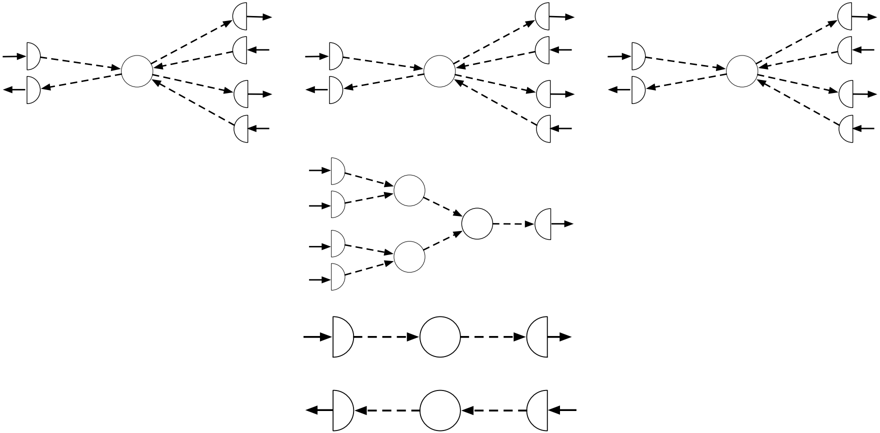


FLICK – the task graph



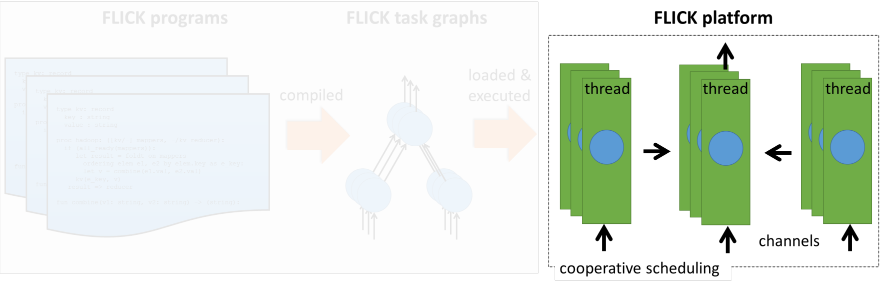
- For memcached router each client has its own task graph.

FLICK – the task graph

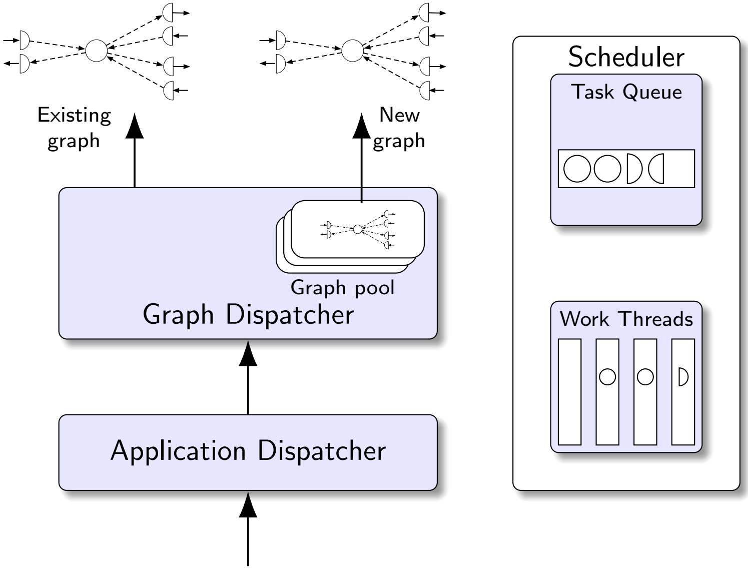


- For memcached router each client has its own task graph.
- Different types of task graph – some have data parallelism.
- Data and task parallelism.

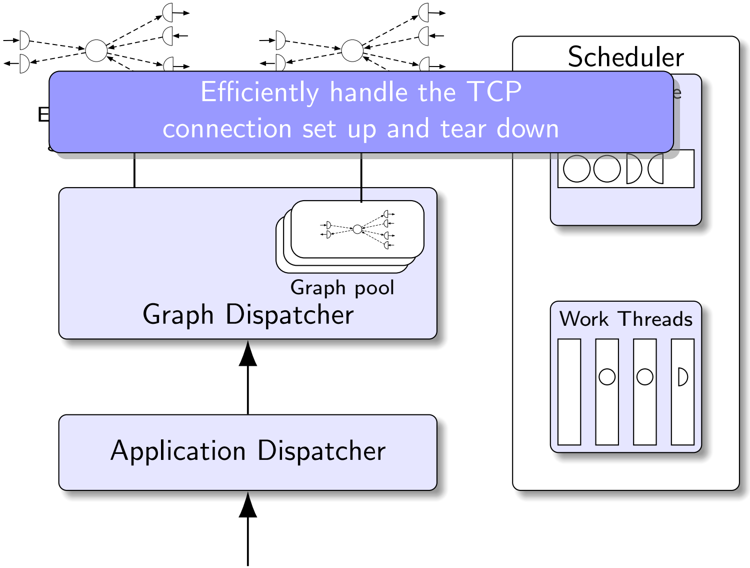
FLICK – the platform



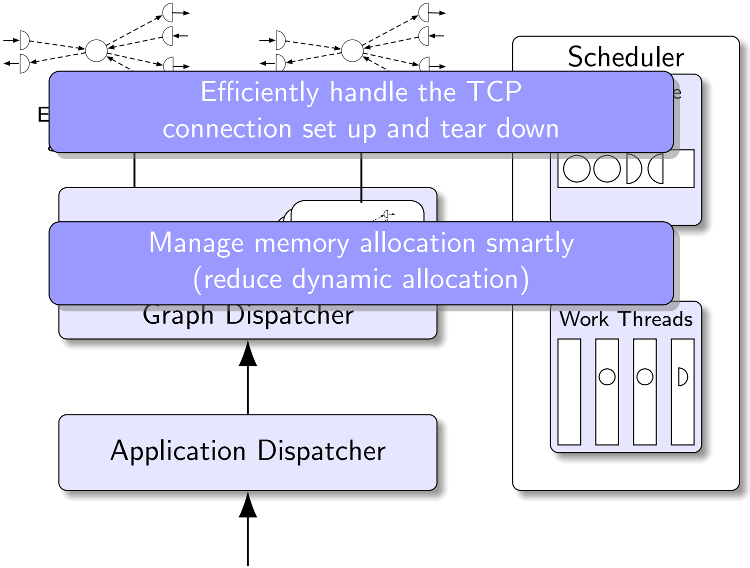
FLICK – the platform



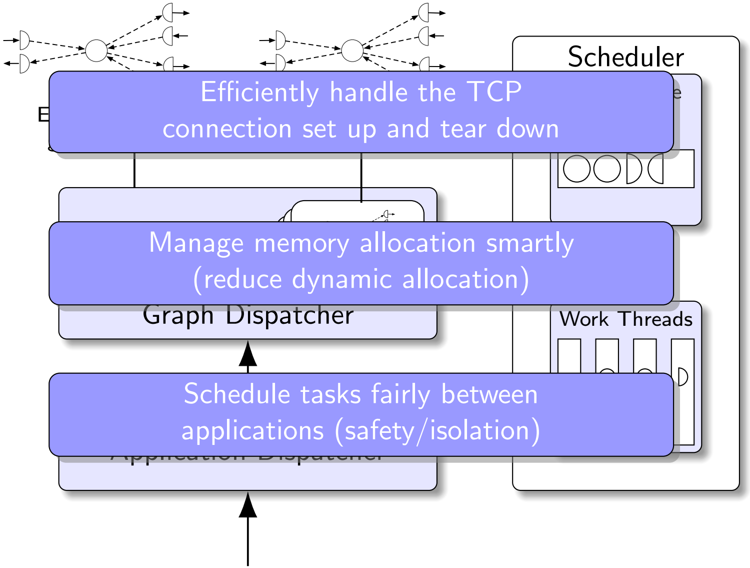
FLICK – the platform



FLICK – the platform



FLICK – the platform

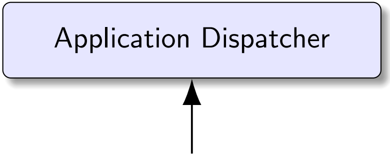


Efficiently handle the TCP connection set up and tear down

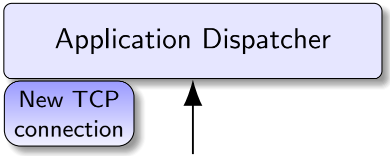
Manage memory allocation smartly (reduce dynamic allocation)

Schedule tasks fairly between applications (safety/isolation)

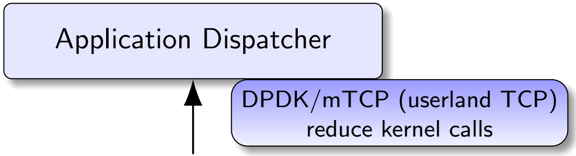
FLICK – the platform



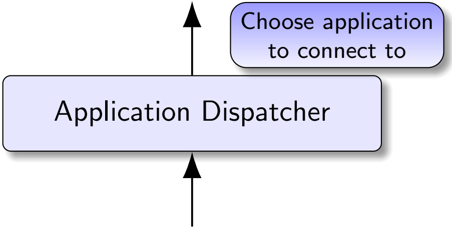
FLICK – the platform



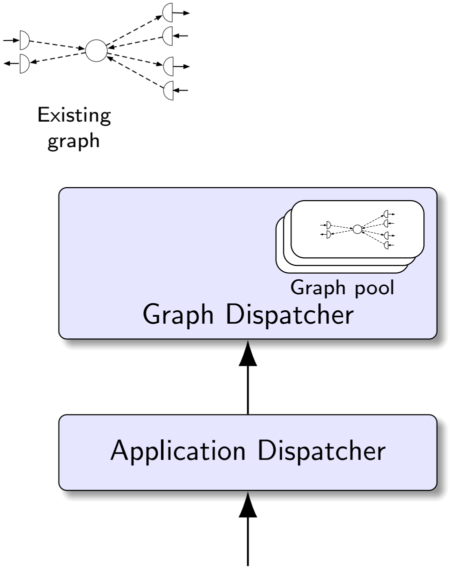
FLICK – the platform



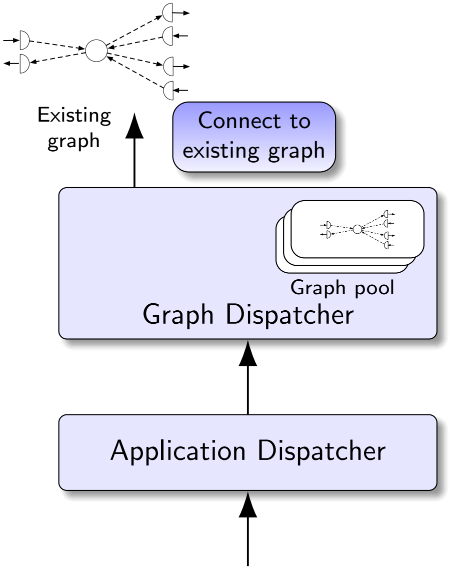
FLICK – the platform



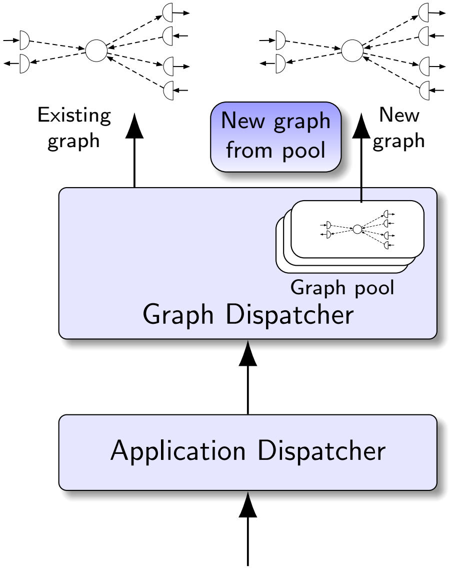
FLICK – the platform



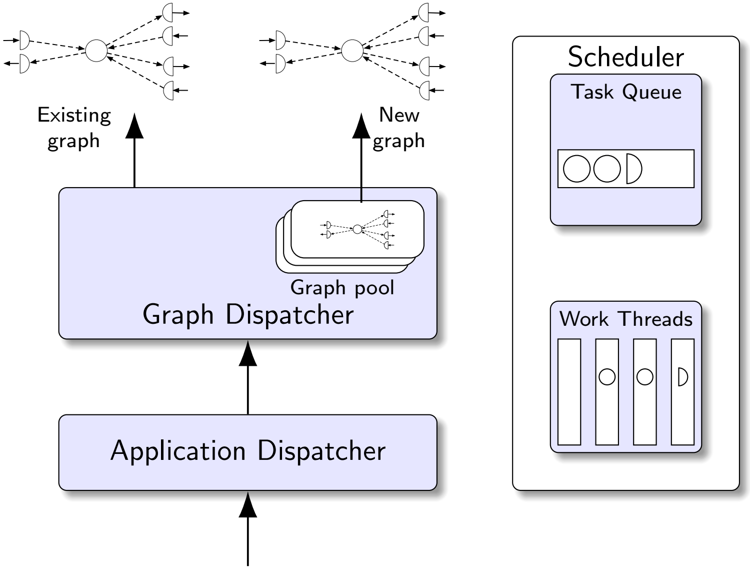
FLICK – the platform



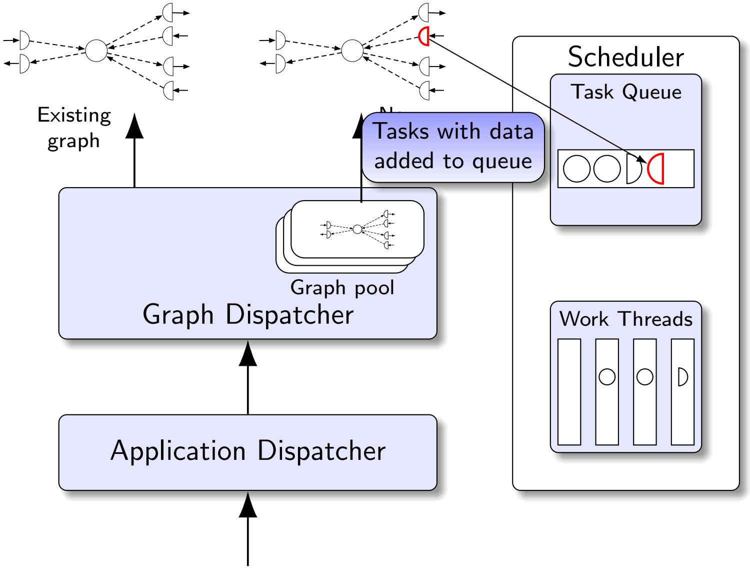
FLICK – the platform



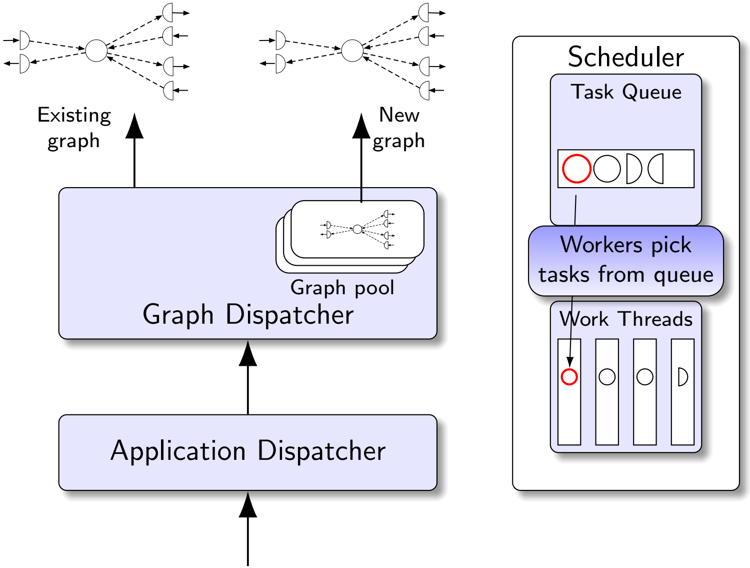
FLICK – the platform



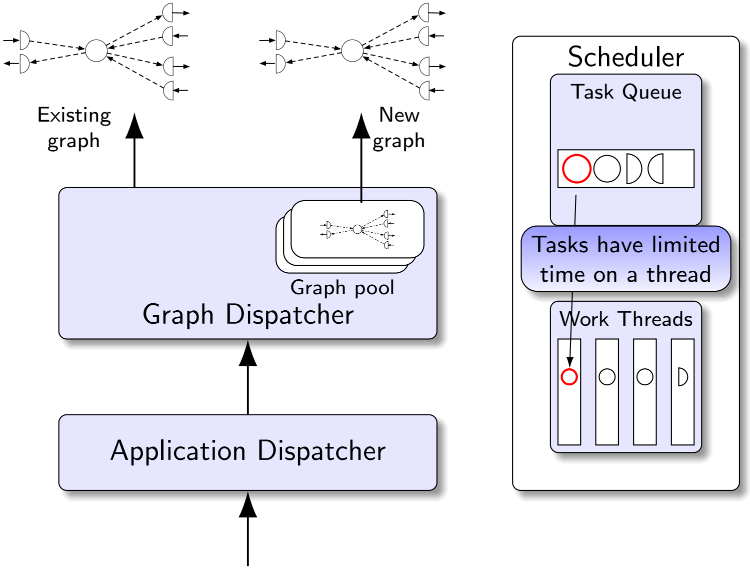
FLICK – the platform



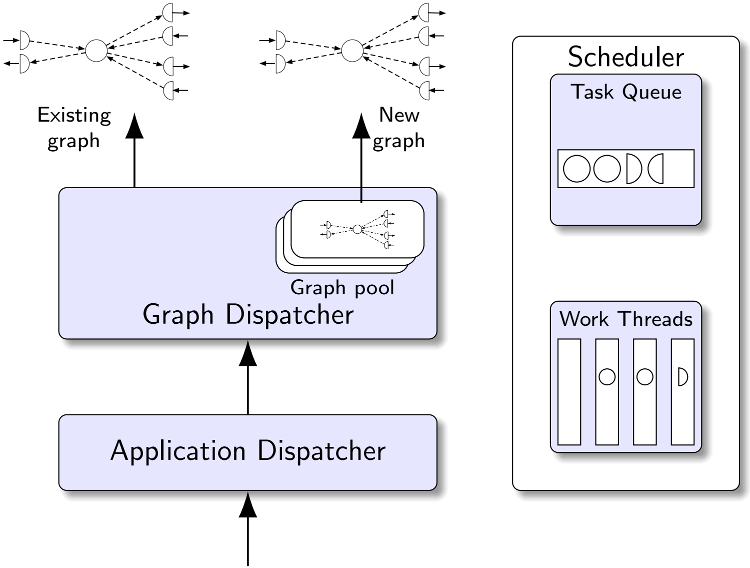
FLICK – the platform



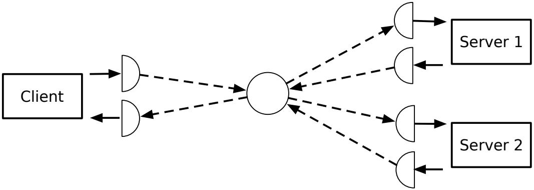
FLICK – the platform



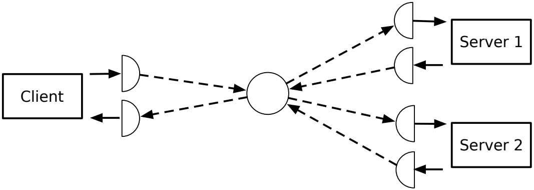
FLICK – the platform



Evaluation – latency/throughput (loadbalancer)

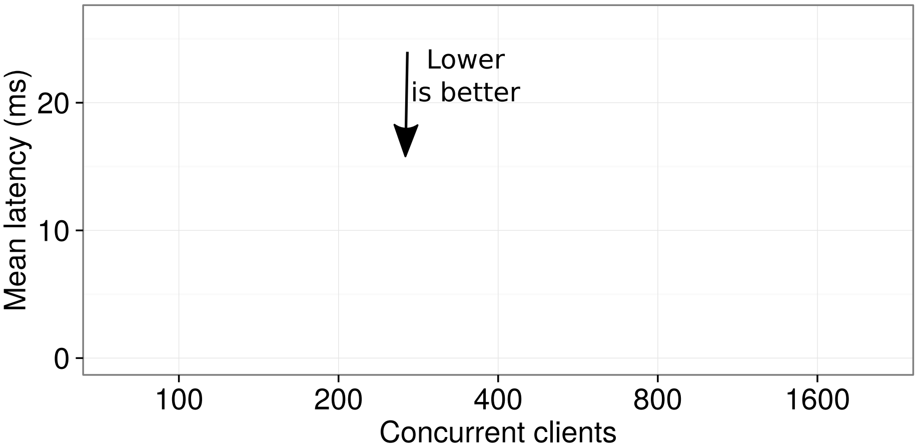


Evaluation – latency/throughput (loadbalancer)

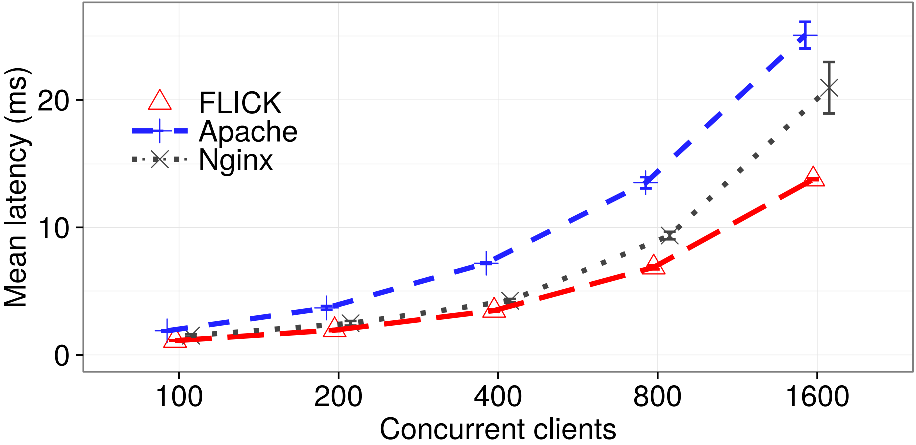


- Clients send HTTP requests up to ten backends.
- Persistent TCP connections to/from loadbalancer.
- Vary number of clients measure latency and throughput.
- DPDK/mTCP used to reduce kernel calls in connections.

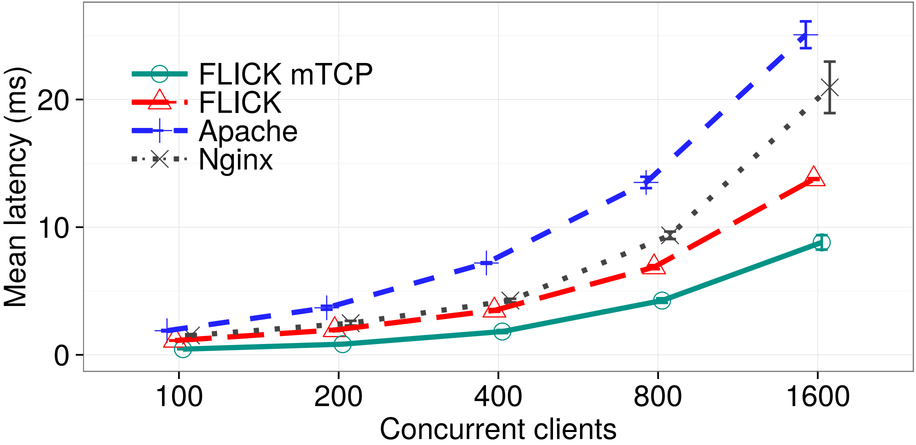
Evaluation – latency (loadbalancer)



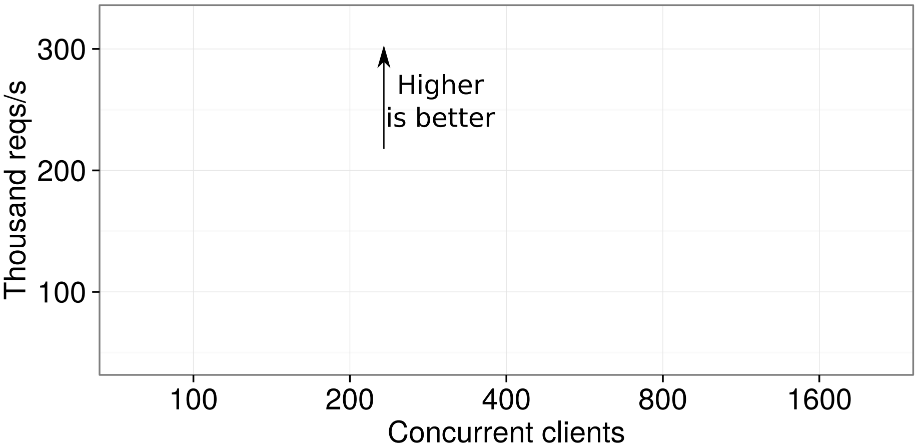
Evaluation – latency (loadbalancer)



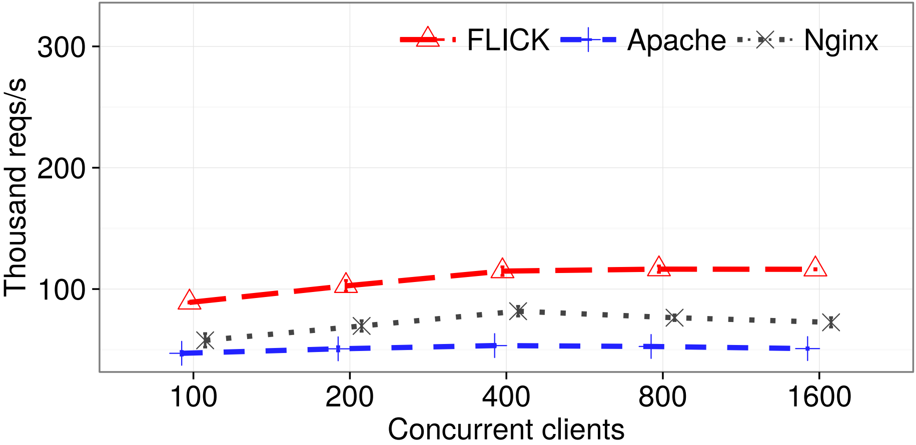
Evaluation – latency (loadbalancer)



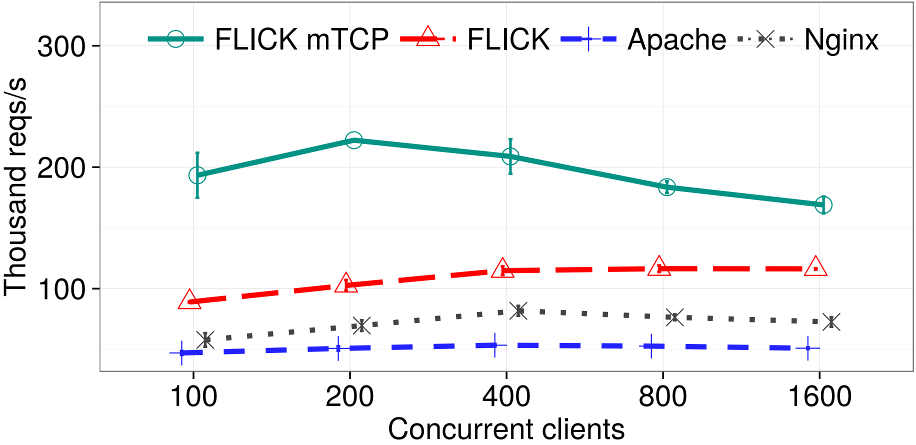
Evaluation – throughput (loadbalancer)



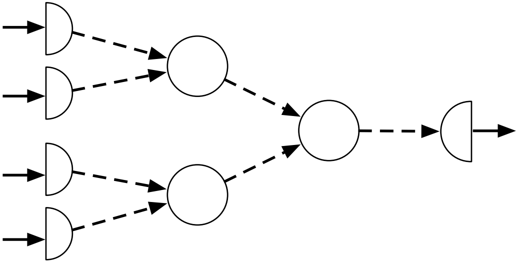
Evaluation – throughput (loadbalancer)



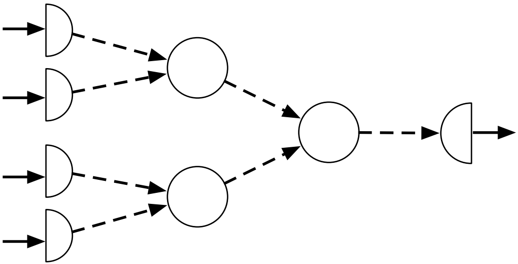
Evaluation – throughput (loadbalancer)



Evaluation – scalability with cores

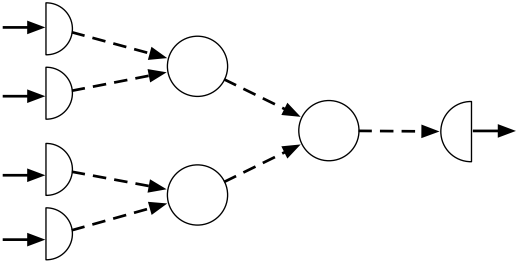


Evaluation – scalability with cores

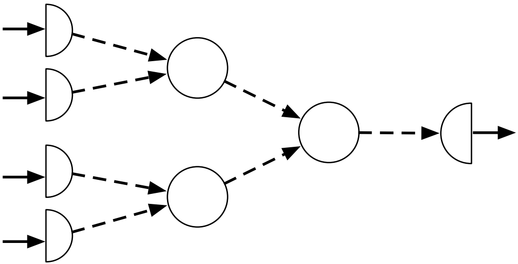


- This middlebox merges data in big data systems.
- Binary merge tree takes advantage of data parallelism.
- See "NetAgg: Using Middleboxes for Application-specific On-path Aggregation in Data Centres" [CoNext 2014].

Evaluation – scalability with cores

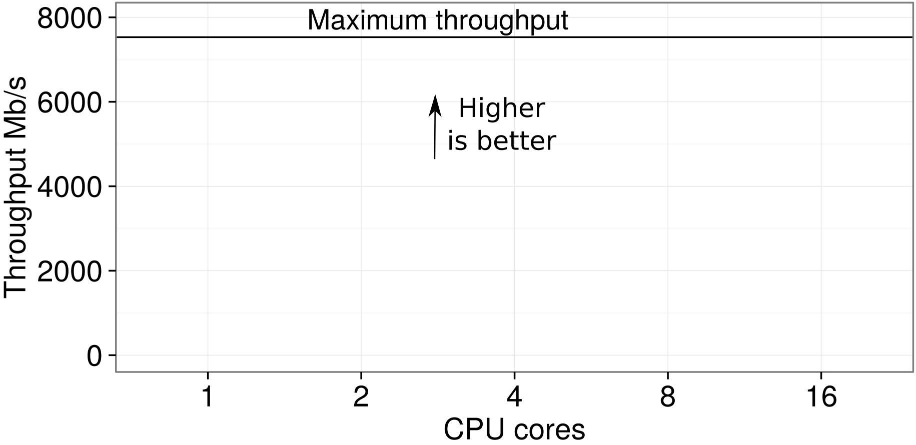


Evaluation – scalability with cores

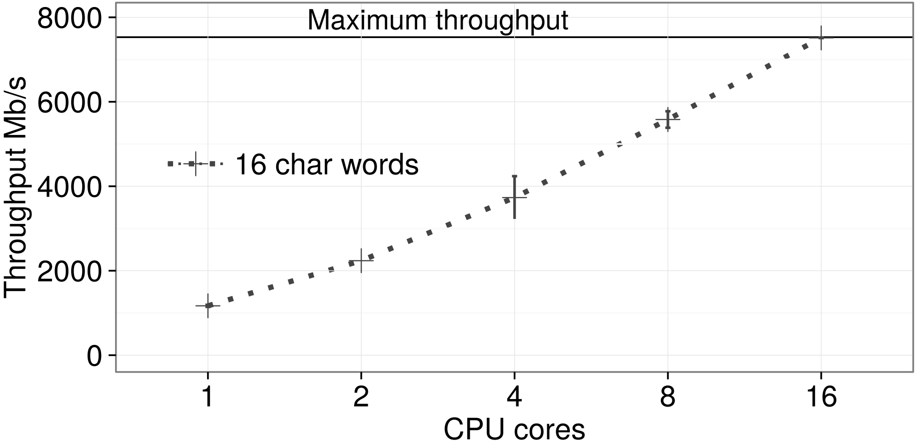


- Test scaling. Measure throughput as number of cores increases.
- Three data sets each one billion words. 8, 12 and 16 character words.
- Merge eight streams – measure throughput of output stream.

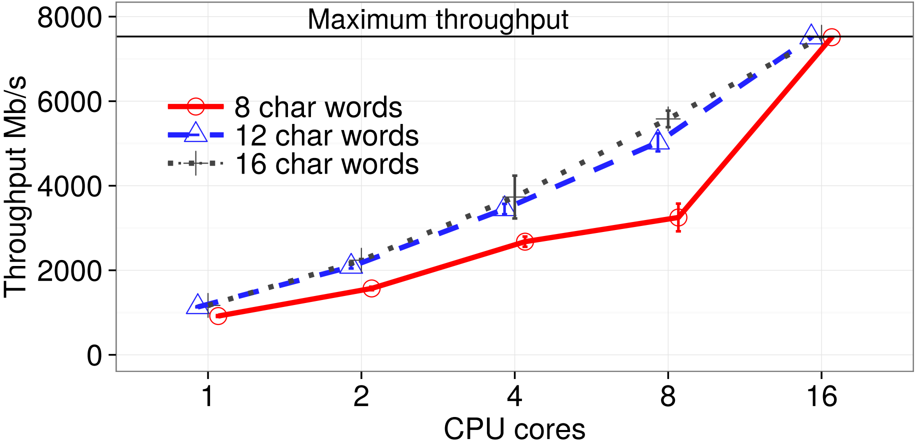
Evaluation – scalability with cores



Evaluation – scalability with cores



Evaluation – scalability with cores



Conclusions

Application-specific services

- Application-specific middleboxes are here to stay.
- Packet processing systems not suitable for these.

Conclusions

Application-specific services

- Application-specific middleboxes are here to stay.
- Packet processing systems not suitable for these.

The FLICK system

- FLICK domain-specific language – “safe by design”.
- Task graph abstraction gives task and data parallelism.
- Performance of FLICK comparable to specialist system.

Conclusions

Application-specific services

- Application-specific middleboxes are here to stay.
- Packet processing systems not suitable for these.

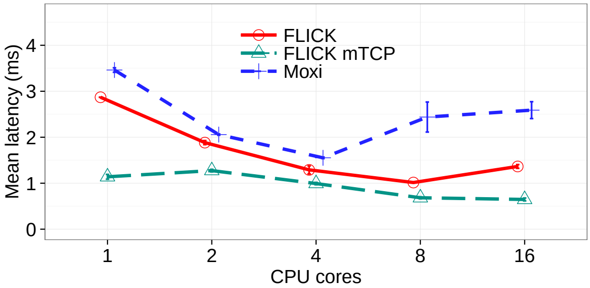
The FLICK system

- FLICK domain-specific language – “safe by design”.
- Task graph abstraction gives task and data parallelism.
- Performance of FLICK comparable to specialist system.

Thank you – questions?

Richard G. Clegg
richard.clegg@imperial.ac.uk

Performance – memcached example



- Comparison with Moxi (also supports multi-core + binary protocol).
- Set up 128 clients making multiple requests.
- Latency reduction shown.
- FLICK throughput with mTCP 198,000 reqs/sec.
- Moxi throughput 82,000 reqs/sec