Distributed CnC for C++

Frank Schlimbach
Intel/SSG/DPD/TPI

2nd annual workshop on Concurrent Collections
Houston, 2010/10/06

CnC for Distributed Systems

- Let CnC utilize scalability of memory/cache-incoherency
- Extend Concurrent Collections to generically support distributed memory
  - KNF (Xn), **Sockets,** MPI, ??
  - combination of the above
- Provide a platform for experiments (proof of concept)
  - Opens another non-trivial dimension of scheduling
  - Can we separate the tuning from the domain?
- Proof-point for abstraction from platform
- **Not** meant to be a general solution for distributed computing

- Minimize extra requirements
  - Minimal incremental changes to existing CnC code
  - Auto/default-partitioning/distribution
  - Keep programming methodology of CnC
- Utilize standard techniques

intel Software

distCnC - Status

- Prototype implementation
- Communication through sockets
- Included in latest what-if release
- Some limitations compared to shared-memory CnC

How to

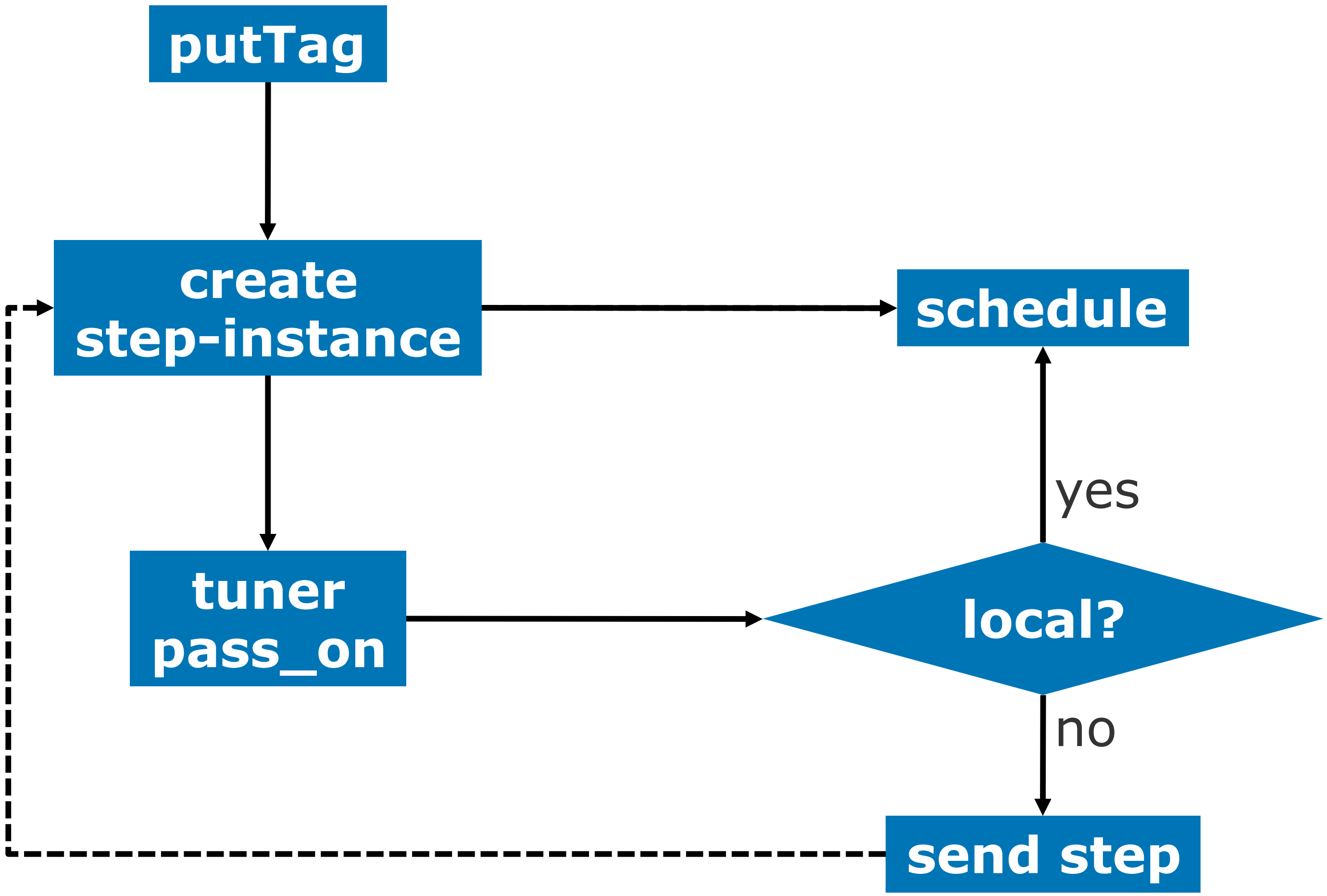
- **#include <cnc/dist_cnc.h>**
  - sets **#define** and declares **dist_cnc_init** template
- instantiate **CnC::dist_cnc_init< … >** object
  - First thing in **main**, must persist throughout **main**
  - Template parameters are the contexts used in the program
- Steps do normal gets and puts

- serialization of non-standard data types
  - Simple mechanism (similar to BOOST)
- The information about where to run a step can be provided by a tuner: **int tuner::pass_on(…)**
  - return process-id for a given tag

- Start up of remote processes through script (or manually)

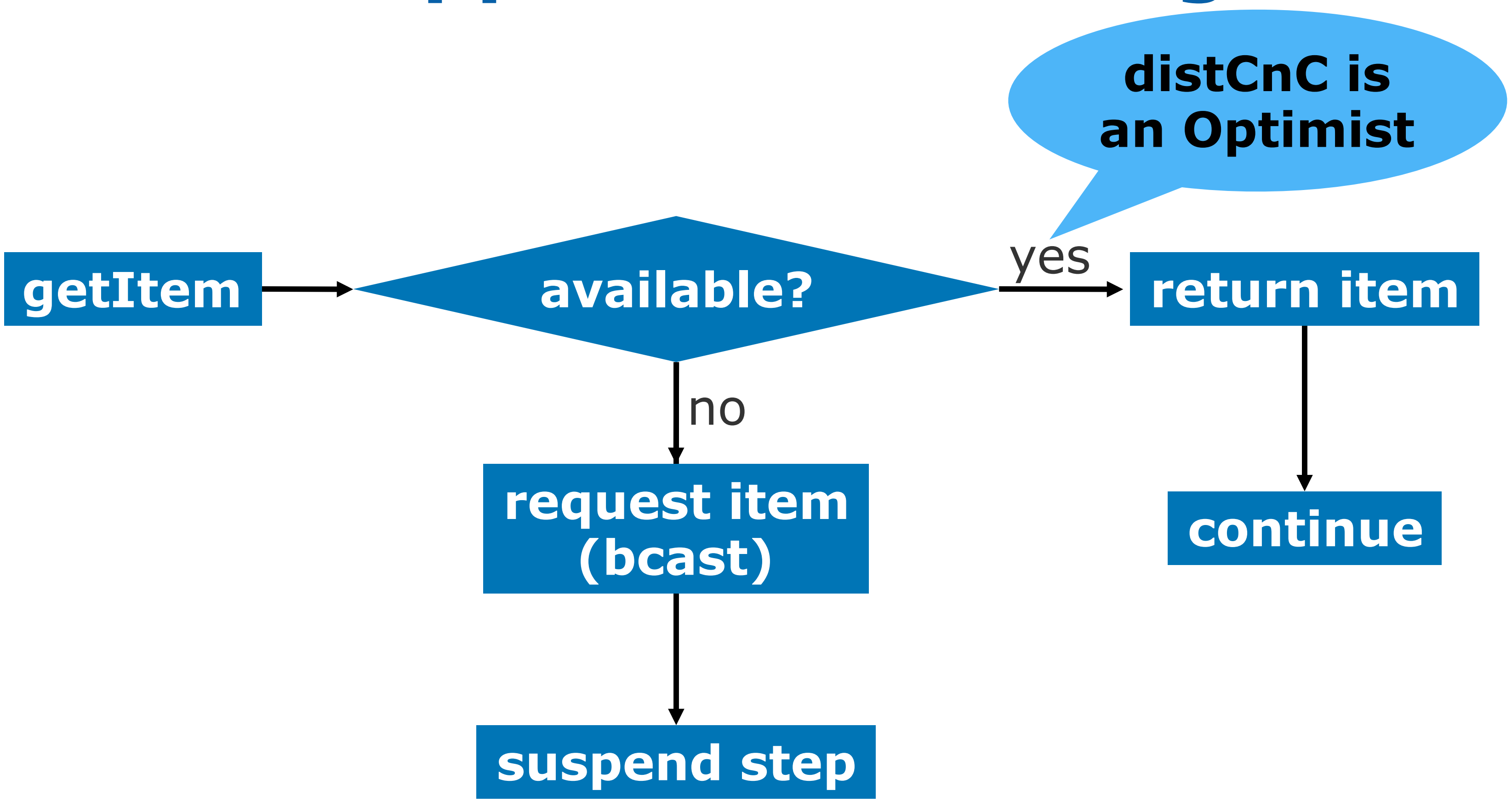
intel Software

What happens in a "tag-put"?

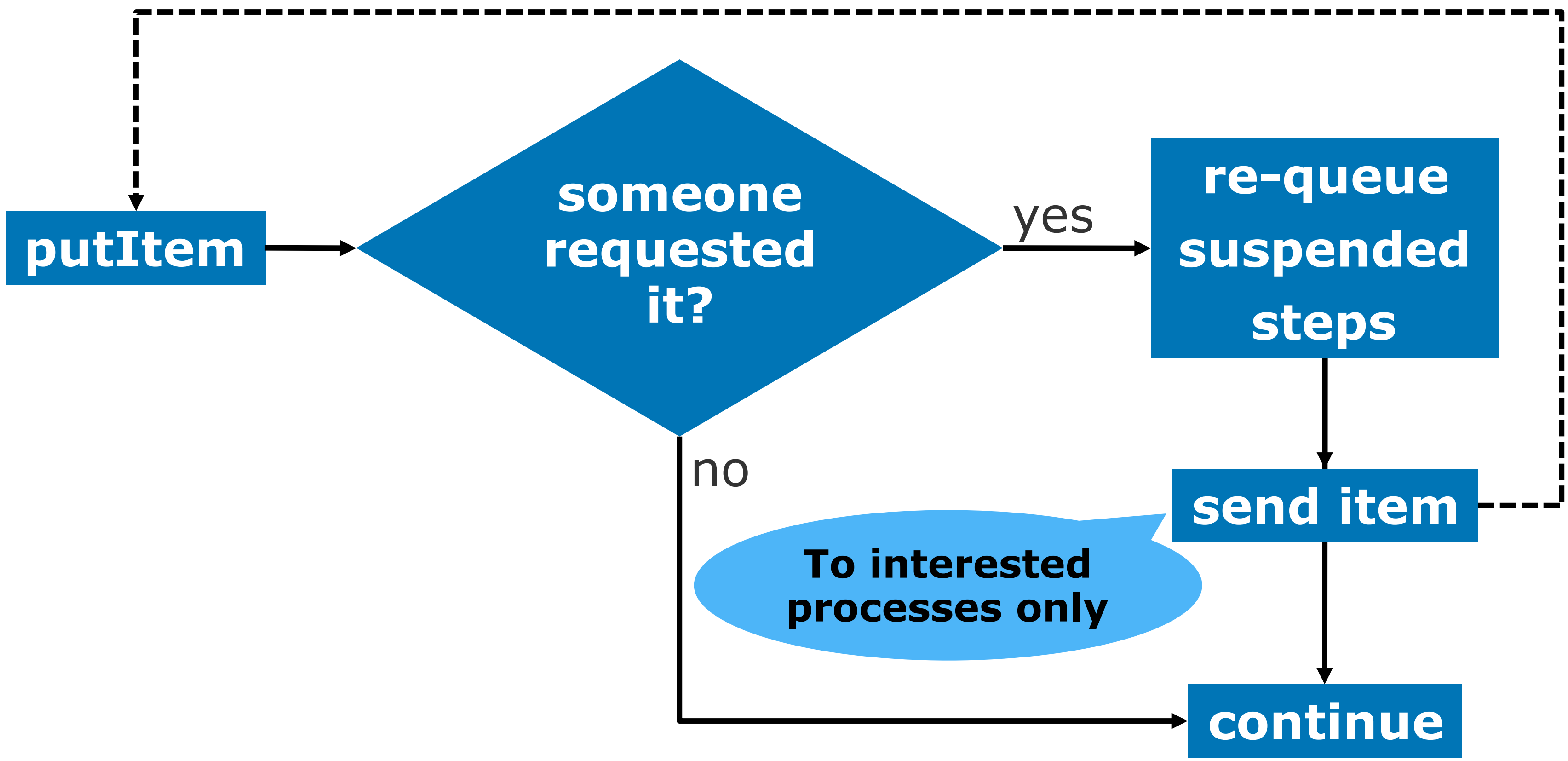


intel Software

What happens in a “item-get”?

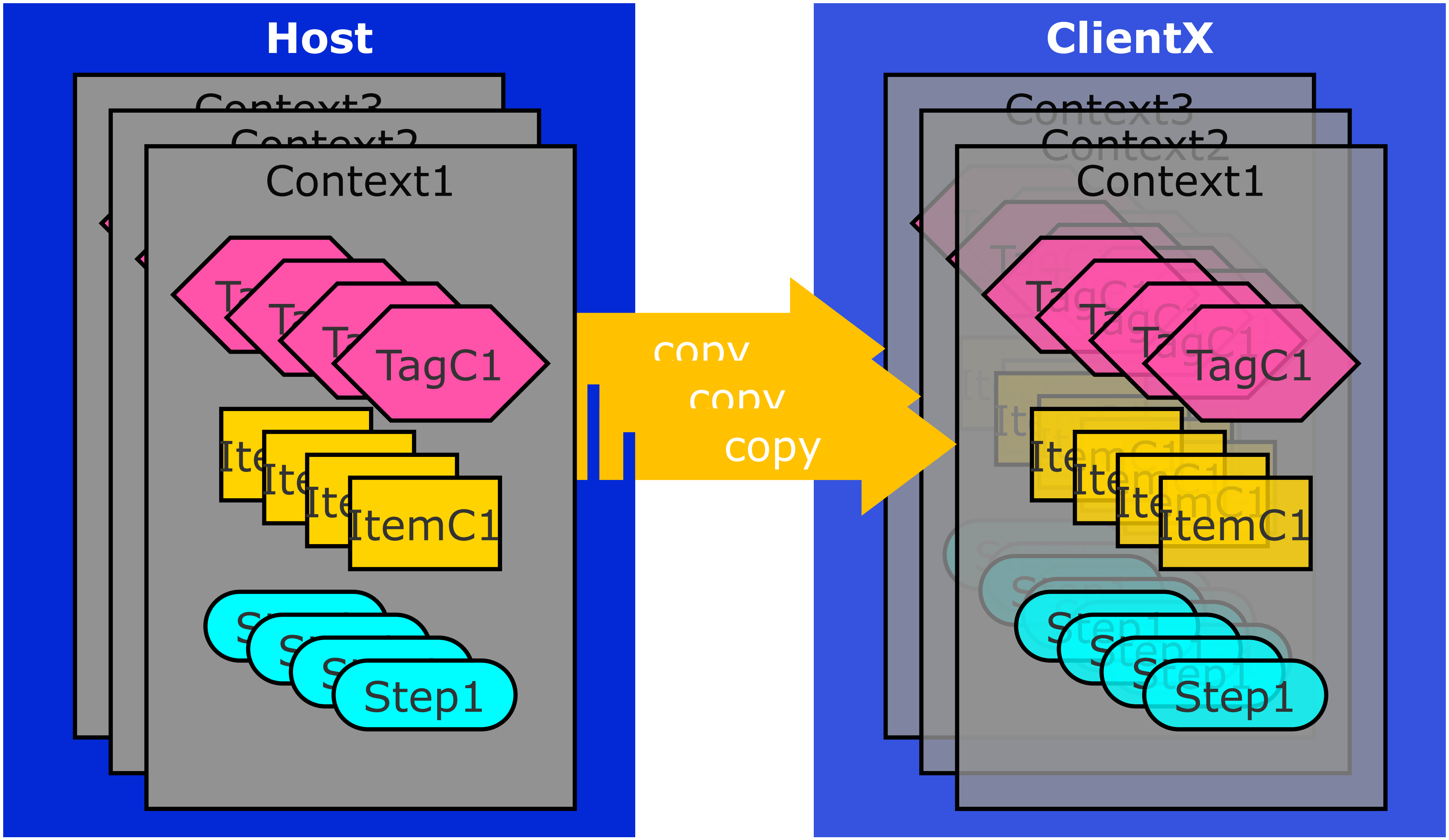


What happens in a "item-put"?



intel Software

Data Residence

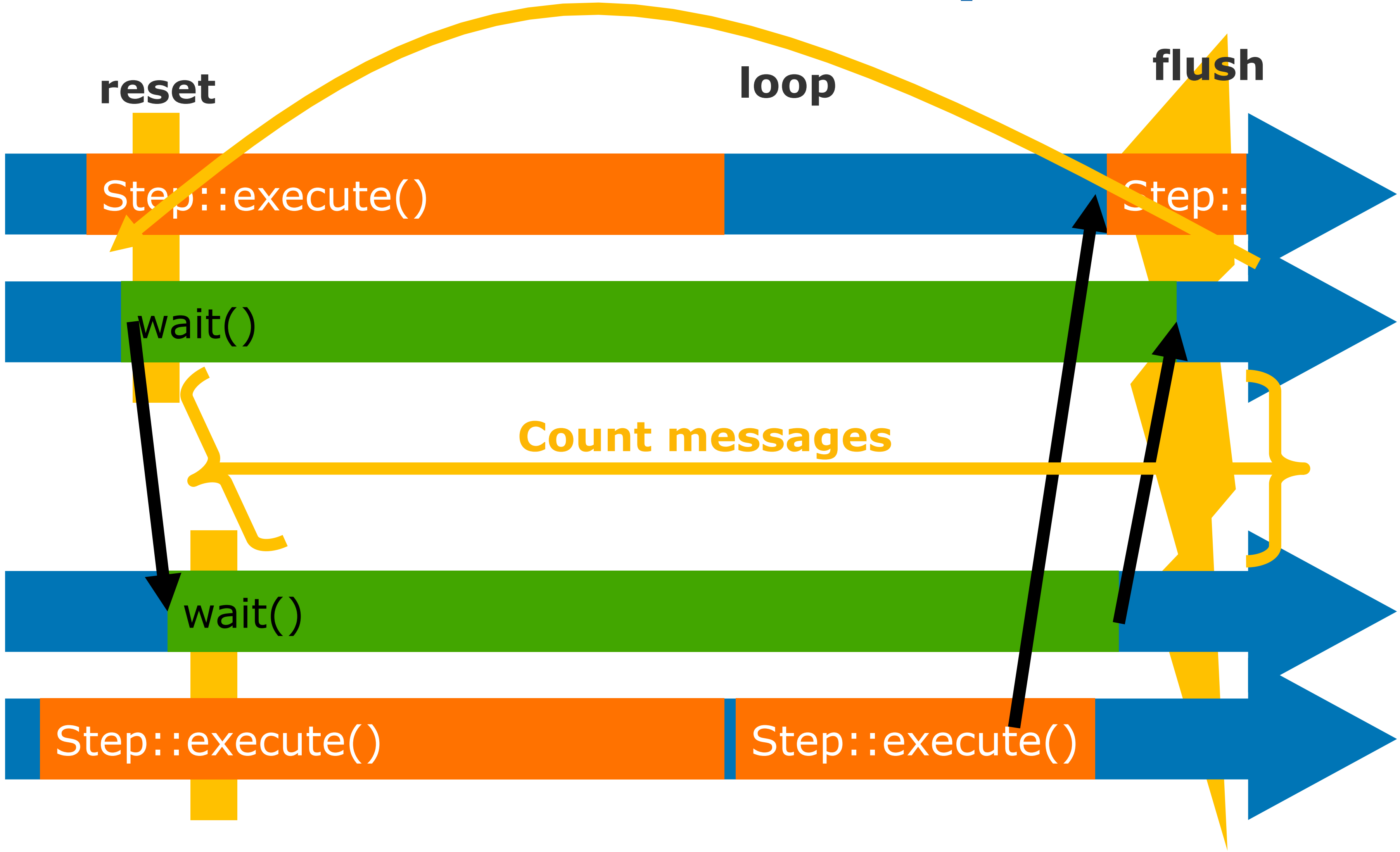


Start up and shut down

- Magic is in **`dist_cnc_init<…>`**
  - Constructor
    - Initializes factory (in charge of creating objects from type-ids)
    - Assigns type-ids to types (contexts only)
      - types of collections are known as they are members of the context
    - Host launches clients, sets up network and continues execution
    - Clients set up network and go into receiver loop they exit when done
      - Clients never leave the constructor!
  - Destructor
    - Host initiates network shut down
    - Clients do nothing

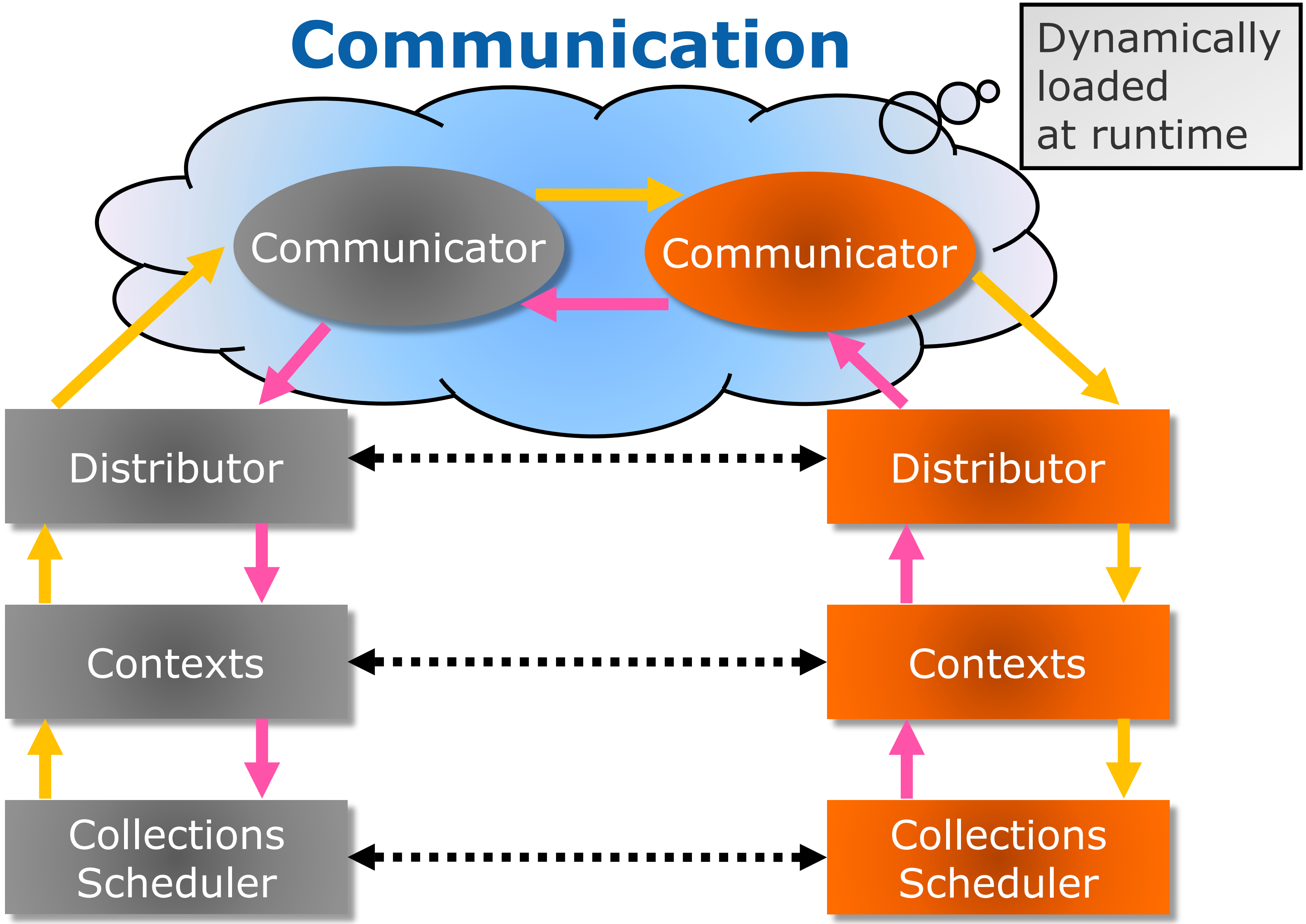
intel Software

Termination detection problem



intel Software

Communication



intel Software

Communicators

- **Sockets**
  - **Loaded at runtime**
  - **Should work across OSes**
- Emulator (incomplete, used to work)
  - Extra thread emulating process
  - requires special linkage
- MPI (incomplete, prototype implemented)
  - Can be done through loading at runtime
  - With MPICH2, nothing could be required
    - Otherwise mpiexec or similar launches the processes
- KNF Xn, native SDK (incomplete, core functionality implemented)
  - Can be done through loading at runtime
  - KNF peculiarities when building the binaries
- System was laid out to allow combining communicators

intel Software

Things to keep in mind

- Collections must be members of contexts (constructed in its ctor)
- Contexts must be default constructible and prescribe steps there
- Tags and items must be default constructible
- Pointers are dangerous
  - Tags must not be of pointer type
  - Items of pointer type need special treatment; better avoid them
- Global variables are evil and must not be used (within the execution scope of steps)
- In contrast to local-only execution, preservation of steps will only locally suppress redundant step execution.
- Tag-ranges cannot be distributed yet, they stay locally

- **All this is aligned with CnC's methodology!**

intel Software

Possible Futures of distCnC

- Performance evaluation
- Alternative communication policies
  - request bundling (lazy)
  - reduce number of broadcasts (user hints, ?)
- Advanced distribution policies
  - Global View
  - Use data about resources (utilization, HW, …)
  - Declare local availability
- Allow distributing ranges (parallel_for)
- User managed data/items/pointers
- Other communicator layers (MPI, Xn, RUDP)?
- Heterogeneous and/or hierarchical networks (e.g. cluster of GPU attached workstations)
- Adding/removing clients on the fly
- Fault tolerance
  - Checkpointing? Continue? Restart (partially)?
  - Failure on client, failure on host

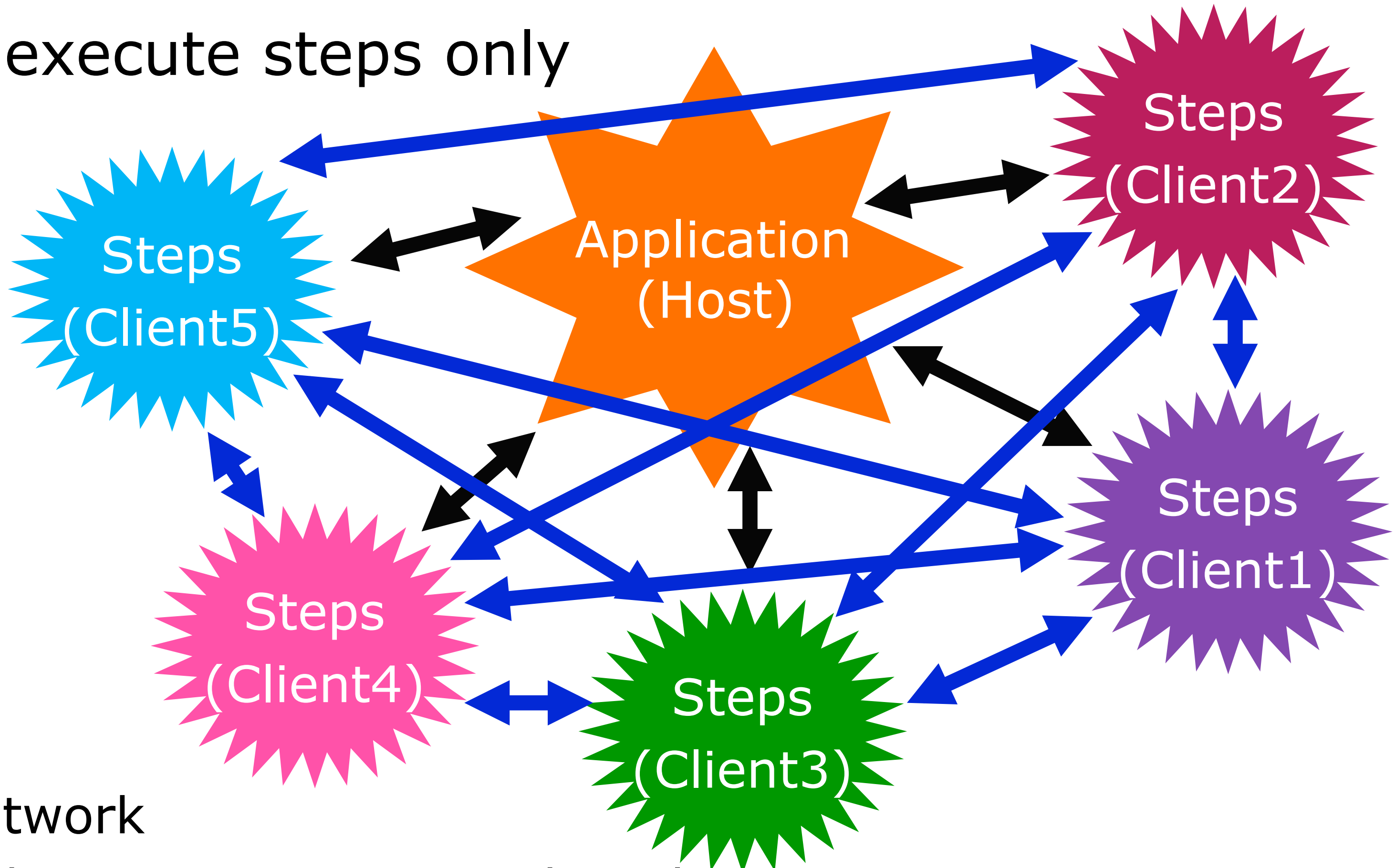
intel Software

intel®

Leap ahead™

Execution philosophy

- Program on host
- Clients execute steps only



- N-to-N network
- Steps might trigger steps on other client processes

Operation

- When a context is created, it is cloned on all clients/processes
  - all its collections will be there automatically
  - context creation creates the scheduler, which creates worker threads
- When a step-instance is created, the scheduler might decide that it must be passed on to another process
- Processes schedule steps upon their reception
- Optimistic execution
  - optimizes for local availability of items
  - if an item is unavailable, it is requested with all other processes (broadcast)
  - if a process has (or creates) requested item, it sends it to those processes which requested it
  - data/item traffic quickly dominates communication costs

Example (quickSort)

```
#include "cnc/dist_cnc.h"
...
void serialize( CnC::serializer & ser )
{
    ser & m_isPartitioned & m_size & m_verbose;
    ser & CnC::array_alloc( m_array, m_size );
}
...
CnC::dist_cnc_init< qs_ctxt > dc_init;
...
struct quick_sort_tuner : public CnC::default_tuner< tag_type, qs_ctxt >
{
     int pass_on( const tag_type & parent, qs_ctxt & ) const
    { return parent % 4; }
};
...
    prescribe( ancestryPathSplitTagSpace,
               quick_sort_split_step(), quick_sort_tuner() );
```

Why Serialization

- Distributed memory systems require serialization for data transfer
- ⇨ Tags and items must be serializable
- C++ language does not provide serialization (like Java or .NET)
- CnC framework provides serialization capabilities which
    - ➢ Make simple things simple
        - ⇨ Built-in serialization of standard data types and ranges
        - ⇨ Array-wrappers with and without memory handling
    - ➢ Make complex things possible
        - ⇨ All data types can be serialized
        - ⇨ Complex structures (e.g. with pointers or virtual methods) require `serialize` method or function
    - ➢ Are easy to use and commonly known (like in Boost)
    - ➢ Do not provide automatism which might fail
        - ⇨ auto-serialization only upon request (simple declaration) compiler issues error if serialization is undeclared

intel Software

Serialization

Bitwise serializable (e.g. structs without pointers; default for builtin types)

`WORKLETS_BITWISE_SERIALIZABLE( MyStruct )`

Explicitly serializable (default)

provide `void serialize( CnC::serializer &, YourType & )`

or `void YourType::serialize( CnC::serializer & )`

one function/method for both serialization and deserialization

very easy syntax, using `operator&` (like in Boost)

```
class MyType {
        int _n;
        float* _arr;
        MySubClass _obj;
    public:
        void serialize( CnC::serializer & buf ) {
            buf & _n;                          // standard data type
            buf & array_alloc( _arr, _n ) // automatic memory allocation
                & _obj;                        // requires its serializability
        }
};
```


intel Software

Launching distributed CnC (sockets)

- On Host, set CNC_SOCKET_HOST
  1. **`number_of_clients`**
  2. **`name_of_script`**

1. Host prints contact string to manually start clients
   `CNC_SOCKET_CLIENT=<contact_string>`
2. Host launches script twice:
   1. **`-n`** must return number of clients
   2. Starting clients with given contact string (e.g. through ssh)

   Example scripts for Windows and Linux are provided

Same executable can be used to run on host and clients; even on a single process without clients.

Debugging and Profiling