1 Master-Slave communication model

Master

- parse Tcl script
- if necessary, communicate: send a request to *all* clients
- decision to communicate depends on script content, order not known
- at compile time - asynchronous mode

Client

- wait for request from master
- initial request content determines further communication - synchronous mode

Requests: set global variables, interactions, particles, start integration,...

2 Global variables / interaction parameters

- Global data object meta information in global.c/global.h: location, type, array size, and callback function.
- callback called only on master node
- on all nodes called: on_parameter_change()
- interactions in interaction_data.c/h via unions
- Unions sent as struct - not portable
- Only global variables, same value on all nodes

3 Particle properties / cellsystems

3.1 Particle data

- data is packed into a single structure, made up from substructures:
  Position, Moment, Force and Properties
- organized into ParticleLists with granular growth
- ParticleList can be communicated synchronously between nodes
- transfer again as structure, not fast
- on the master node, functions exist for simple modification of particle properties
- particle properties can be added by simply hacking into the structures

Main disadvantage:

- huge cache overhead: cache loads always 32 byte, but position / velocity is 24 bytes → loading 64 bytes per particle, using only 48
- presumably a speed difference of factor around 2-3
- solution: separate lists for each particle property
- adding a particle property by hacking gets difficult without precompiler

3.2 Cellsystems

- Already limited OO with communication descriptors and callbacks for particle positioning
- on the other hand lots of switches
- Hacking a cellsystem is *demanding*
- Cellsystems rely on a master and are symmetrical over nodes - no load balancing, adaptive coarse graining efficiently possible

3.3 Ghost communication

- Holy grail of Espresso
- Four communications: ghostcell size update, full ghost particle exchange, ghost position update, ghost force collection: GhostCommunicators
- Each collection is a list of communication commands specific for each node: GhostCommunication
- Each communication has an operation and a list of cells to operate on
- In addition, prefetch and poststore bridge the delay to to building the comm buffers.

4 Espresso++

4.1 Why rewrite?

- Object orientation - improves extensibility
- precompiler - meta objects
- clean up of grown concepts
- extensibility now also for integration concepts etc
- algorithmic details can survive (potentials, ELC, P3M???)

4.2 Precompiler

- like Qt's moc
- is stable
- code contains additional declaration macros
- code contains new loop constructs: loops over meta information
- meta loops disappear at compile time, no performance impact
- allows much more unification - adding a new feature doesn't have to be holistic

4.3 C++

- we already have some OO concepts
- less side effects
- Encapsulation allows better exchangability of objects, integrators, cellsystems etc.

4.4 Limitations of extensibility

- communication limited to MPI - not necessarily available on future hardware
- cellsystem are not flexible: no load balancing etc.
- No meta integrator concepts, such as Parallel tempering, FFS, ACG
- adding particle properties is not easy, adding cellsystems nearly impossible