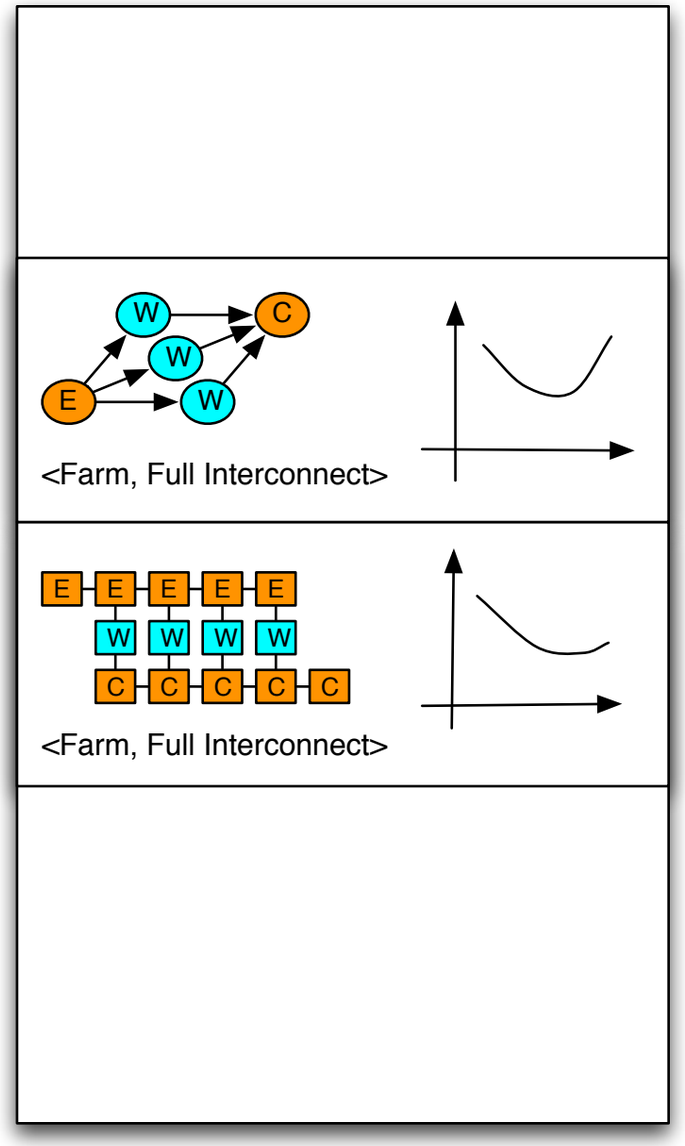
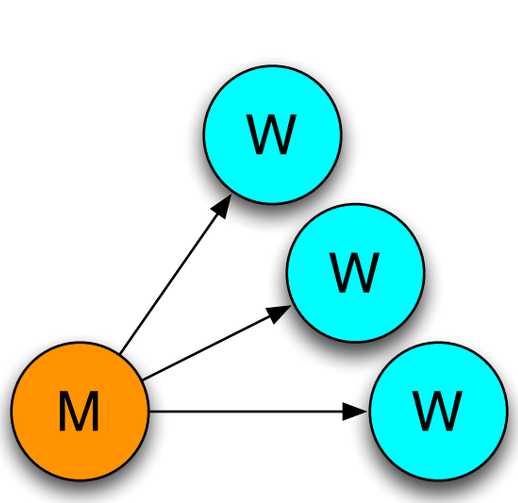
Template libraries

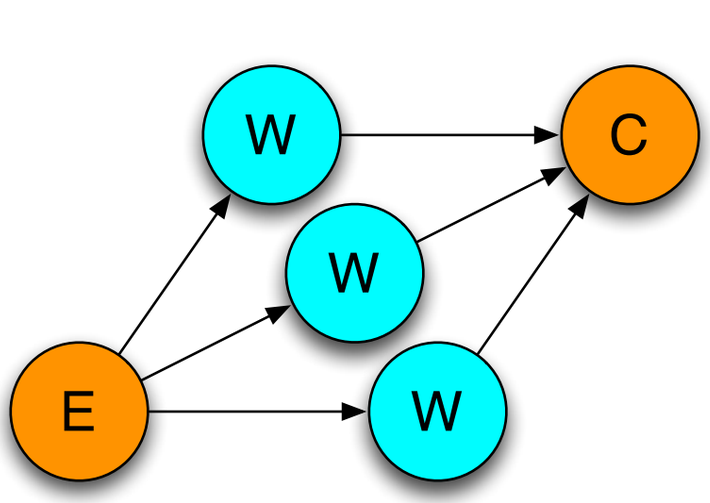
- **Multiple entries per <skeleton,target>**
  - *choice depending on the performance models*
  - *and/or heuristics (possibly programmer driven)*
- **Each template**
  - *exploits knowledge on the skeleton,target pair*
  - *may derive from different experiences*
  - *may optimize different measures*

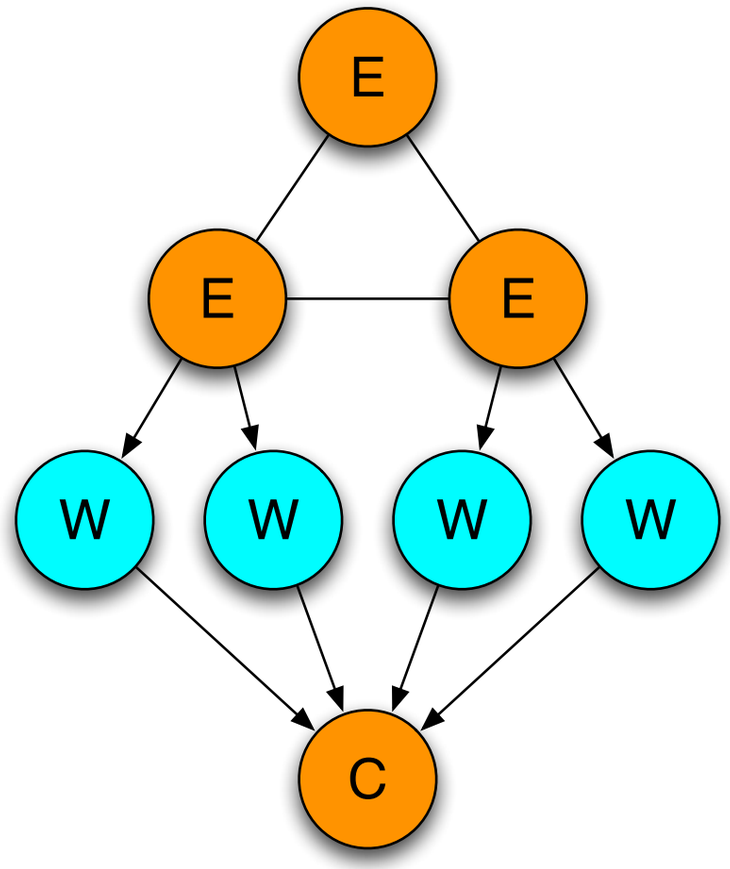


E.g. Sample template library entries

- **<Master/Worker, fully interconnected PEs>**
  - *increasing bandwidth (assume results are more compact than tasks)*







Template based frameworks

- **eSkel (Edimburgh (UK), Murray Cole, C/MPI)**
  - *two versions, farm and pipeline supported + implicit/explicit comms*
- **Muesli (Muenster (D), Herbert Kuchen, C++/MPI)**
  - *high level usage of C++ templates*
- **SkeTo (Tokio (J), Matsukaki, C/MPI)**
  - *data parallel library, intended for seq programmers (à la TLB)*
- **P3L (Pisa, Danelutto+Pelagatti, C/MPI)**
  - ➙ *SkIE (Pisa, QSW, C+F77+Java/MPI)*
- **ASSIST (Pisa, Vanneschi, C POSIX/TCP)**
  - *introduces "generic" skeleton* ***parmod***

Phases

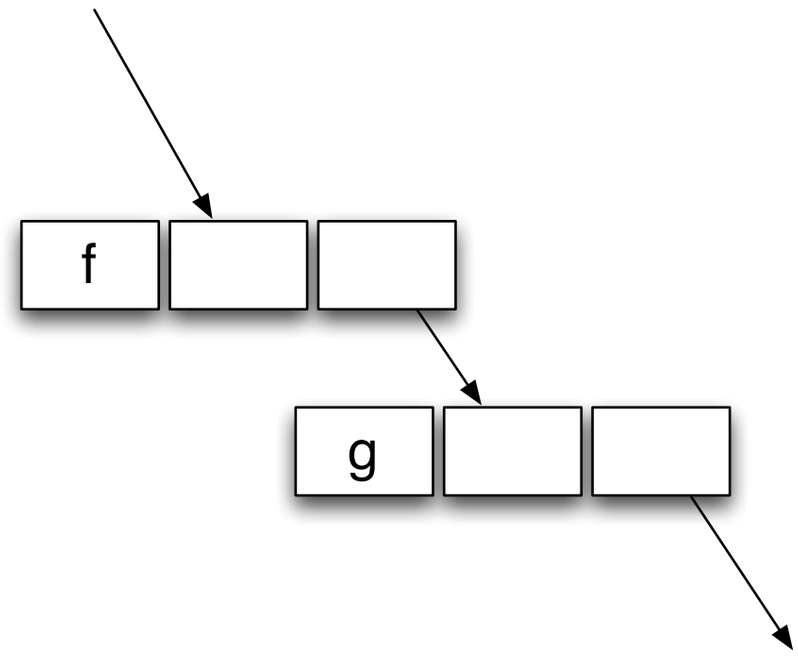
- **Compile time**
  - *decide how to compile + generate parametric code*
- **Deploy time**
  - *compiled program is transferred to remote nodes + parameter adjustment (e.g. number of workers)*
- **Execution time**
  - *run time. Here only the things programmed in the templates happen*

Macro data flow implementation

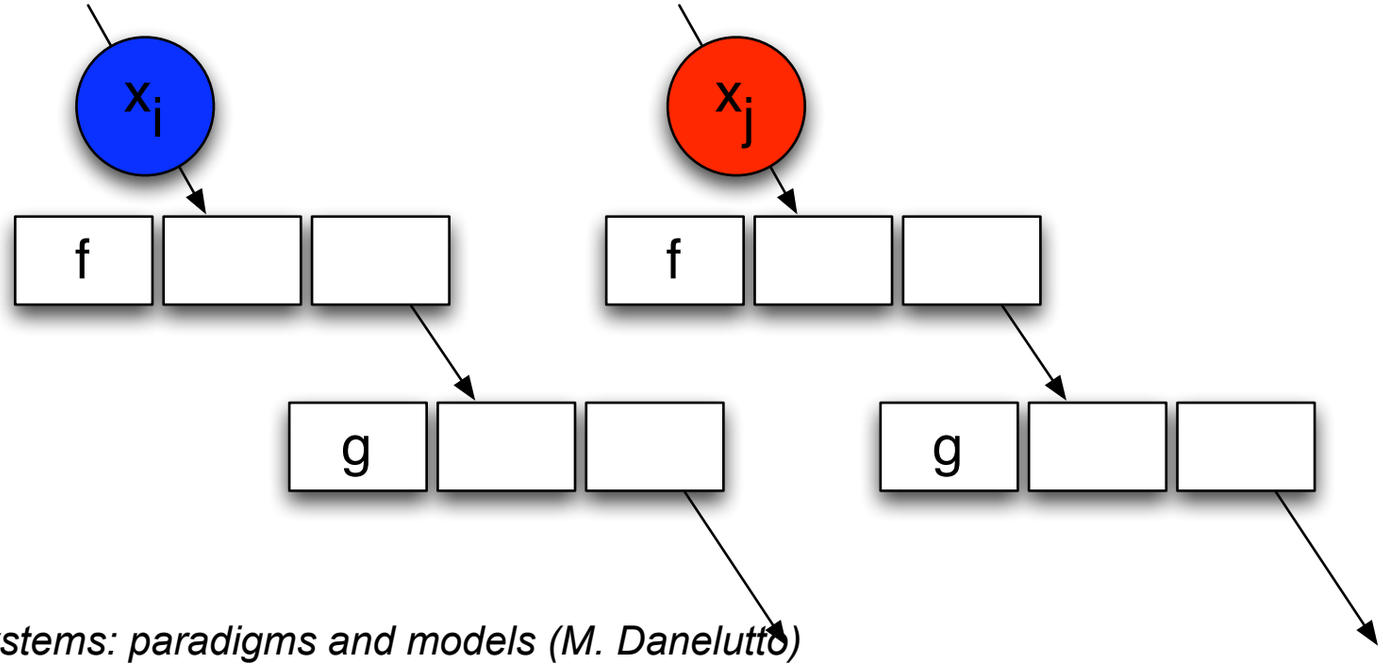
- **New technique introduced in late ’90**
- **Main concept:**
  - *each skeleton corresponds to a data flow graph*
  - *skeleton program is compiler to a data flow graph*
  - *each new task on input instantiated a copy of the graph*
  - *graph copies are “reduced” via a distributed data flow graph interpreter*

Macro data flow idea

- pgm = pipeline(seq(f), seq(g))

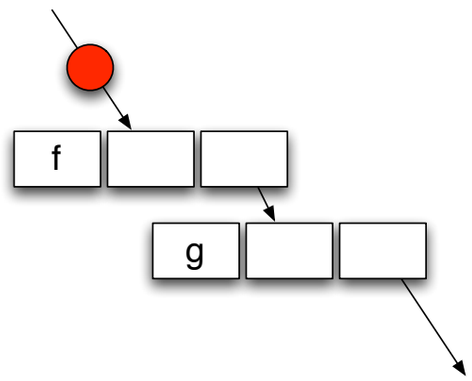


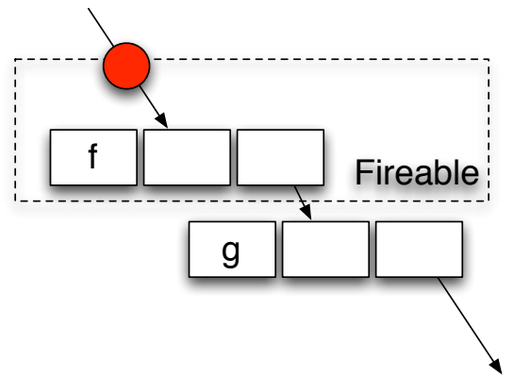
- data flow graph
- instantiation

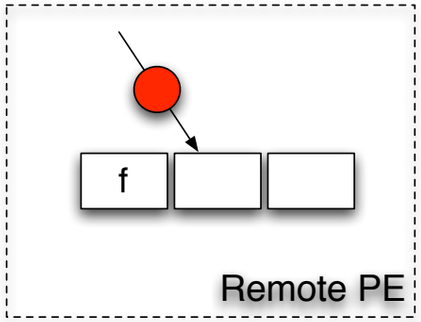


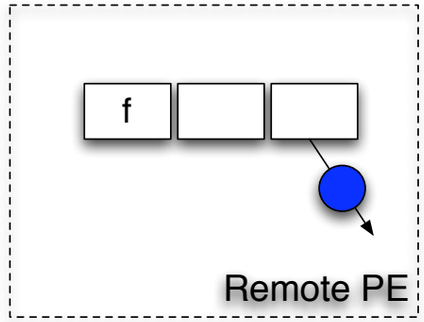
Macro data flow: execution

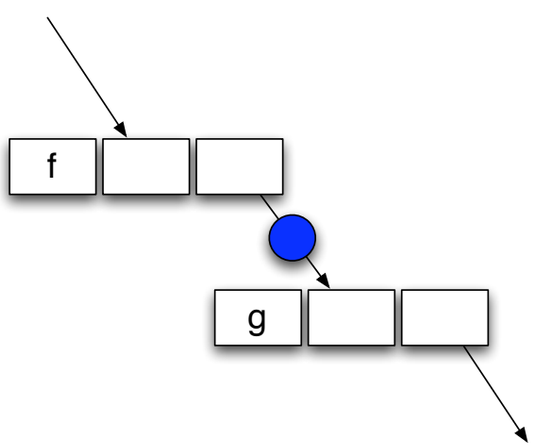
- **Fireable instructions**
  - *selected for exec*
  - *sent to remote PE*
- **Remote PE computes instruction & sends back result token**
- **Result token is stored in the proper instruction**
- **Loop**

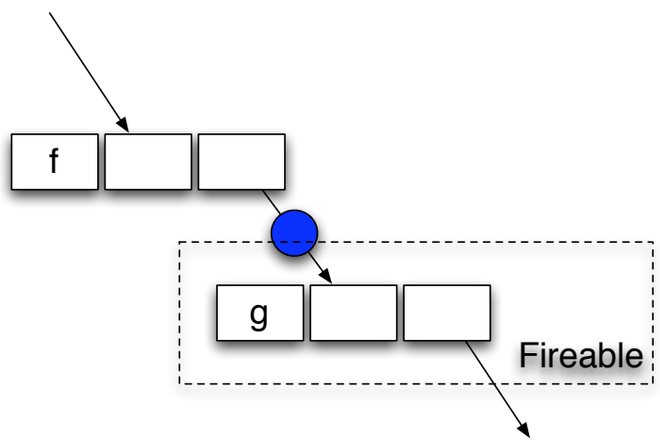






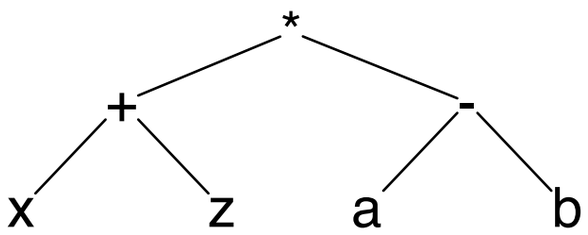


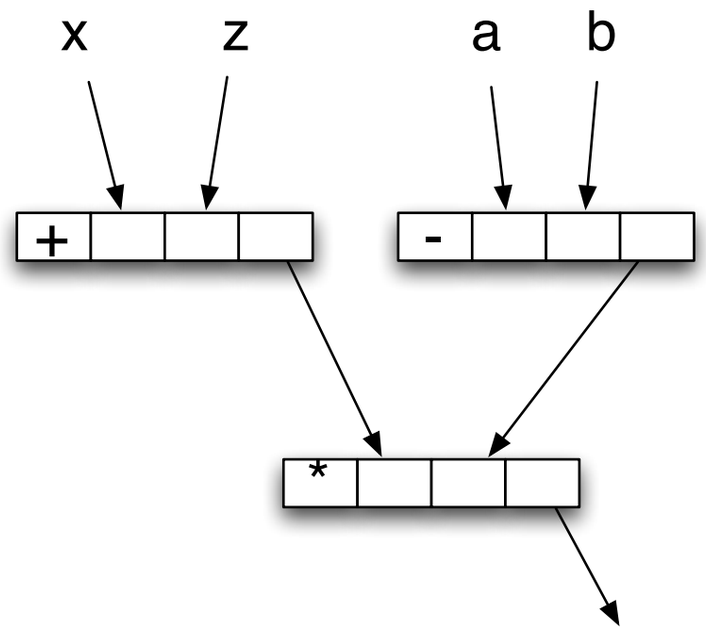




Macro data flow

- **why “macro”**
  - *usually data flow used to manage expression evaluation*
  - *instructions are simple in that case*
  - *here they may be entire portions of host code*





Macro data flow: PROS

- **MDF Instructions can be executed at any site**
- **simpler compiler / run time tools**
- **automatic load balancing**
- **normal form optimization is still feasible**
- **simpler fault handling**
  - *due to anonymous remote MDF interpreters*
- **general purpose interpreter**
  - *can be used to extend “on-the-fly” the skeleton set*

Macro data flow: CONS

- **Macro data flow instruction code needed at all remote interpreters**
  - *serialization vs. source code transfer + compiler*
- **Sequences of instructions with common data tokens require traffic with the host**
  - *g(f(x)) ➙ x to PE(f) ➙ f(x) to Master ➙ f(x) to PE(g)*
  - *if PE(f) coincides with PE(g) ➙ useless traffic*
  - *can be alleviated using*
    - affinity scheduling + token caching

Affinity scheduling

- **MDF instructions accessing the same input tokens**
  - *scheduled to the same (set of) nodes, if possible*
- **requires more complex “master” code**
  - *fireable instruction picked up not randomly*
  - *meta information on previously scheduled MDFi should be maintained at the scheduler node*

Token caching

- **Data flow analysis**
  - *individuate “reusable” tokens*
- **Then**
  - *sent tokens to remote interpreters with a “to cache” mark*
  - *sent further instructions using these tokens with token references rather than token values*
- **Remote interpreters**
  - *keep “to cache” tokens in a local token cache*
  - *when getting a token reference, if not in cache, ask token to master (double traffic on cache miss)*
  - *only worth for large tokens*

Macro data flow based frameworks

- **Lithium → muskel (Pisa, M. Danelutto, Java/RMI library)**
  - *master thesis (Lithium, P. Teti)*
  - *“light” experimental framework (muskel)*
- **Skipper (Clermond Ferrand, Jocelyn Serot, Ocaml library)**
  - *image processing*
  - *several versions, recently moving to C/C++ implementation*
- **Calcium (Sophia Antipolis (F), Mario Leyton, ProActive/Java library)**
  - *not exactly MDF, although most of the concepts preserved*
  - *execution of stacks rather than MDFi from MDF graphs*