Reliability in the Mojave Compiler

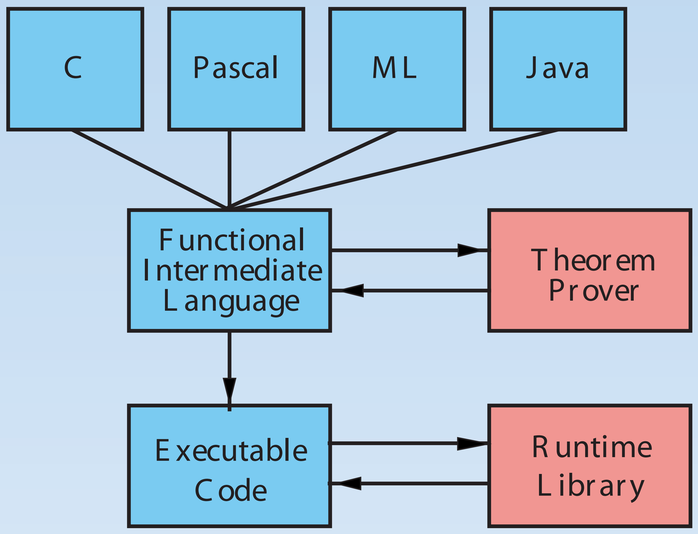
Justin Smith, Jason Hickey, Nathaniel Gray, Adam Granicz, Brian Aydemir
Computer Science Department, California Institute of Technology

Mojave

Mojave Compiler Design

Mojave Compiler (MCC) uses a Functional Intermediate Language (FIR) to provide type safety and formal semantics for many source languages (C, Pascal, ML, Java).

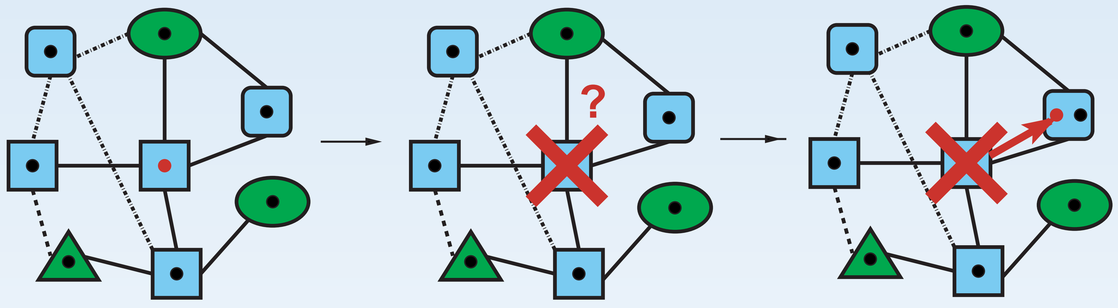
Formal methods allow us to verify all values are treated with appropriate types; some runtime safety checks are required for properties that cannot be verified in the FIR (such as array bounds check).



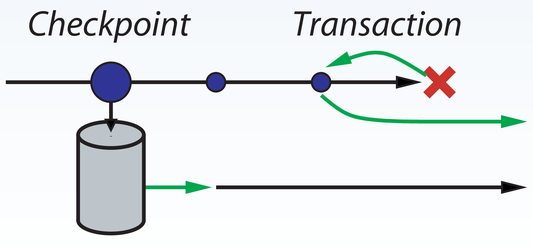
Safety means a process will not crash the machine due to an invalid memory access or any other invalid operation.

Fault Tolerance using Migration

MCC provides mechanisms for fault tolerance, in the form of process **migration** and **transactions**.



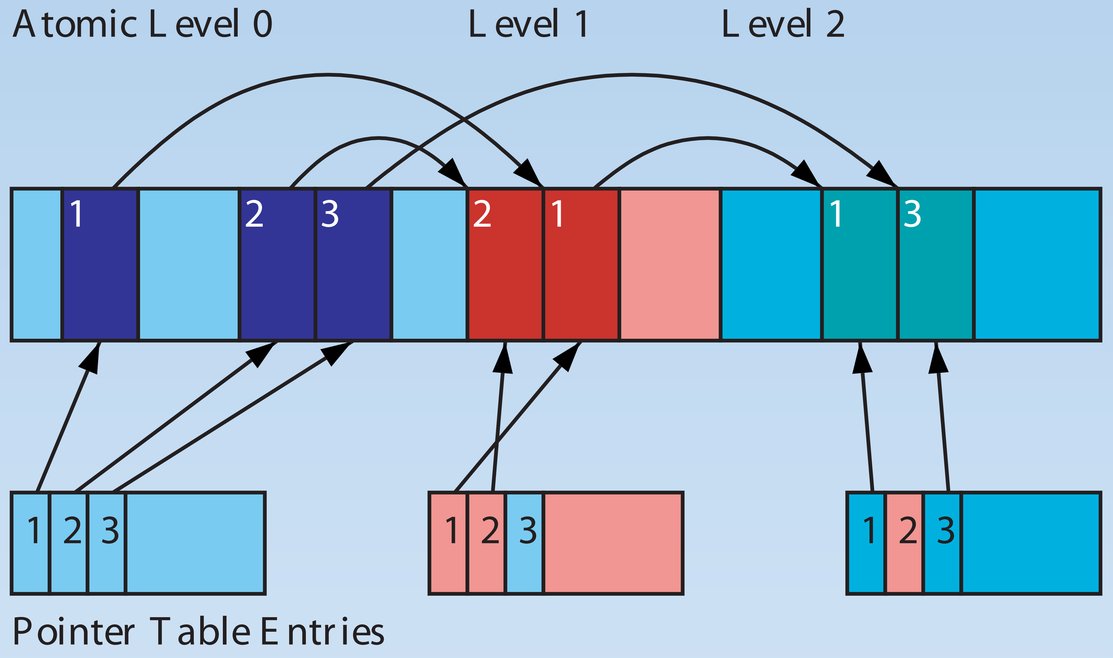
Migration allows a process to save checkpoints periodically; if the machine the process is running on fails, the process can be resurrected on another machine using the checkpoint.



Checkpoints are expensive: for processes that survive a failure, we want a faster way to revert their state.

Fault Tolerance using Transactions

Transactions allow processes in a distributed computation to rollback the computation to a previous valid state if a failure occurs.

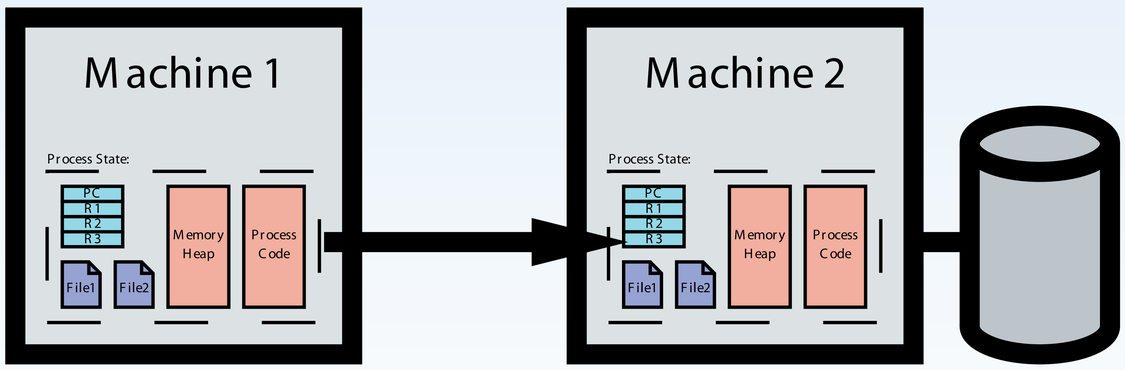


Runtime uses garbage collector to store memory values; the garbage collector is divided into multiple generations, one generation for each transaction. Data in older generations is preserved using **copy-on-write** semantics.

How a programmer uses Fault Tolerance

```
while(true) {
    atomic_begin();  // Enter synchronous transaction
    compute();       // Process computation
    atomic_end();    // Commit the transaction
}
```

Other Applications for Migration



The process is no longer tied to a particular machine; Migration allows for mobile agents which utilize machine transparency for load balancing, resource localization.