Отказоустойчивость в системах параллельного программирования

Беляев Н.А.

Введение

- Рост числа узлов суперкомпьютеров
- Рост сложности организации сетей
- Важным критерием является время реакции на отказ

Подходы к обеспечению отказоустойчивости

- Модификация алгоритмов
- Специальные системы параллельного программирования
    - Checkpoint Restart
    - Let it fail
    - Tasks migration

Отказоустойчивые системы параллельного программирования

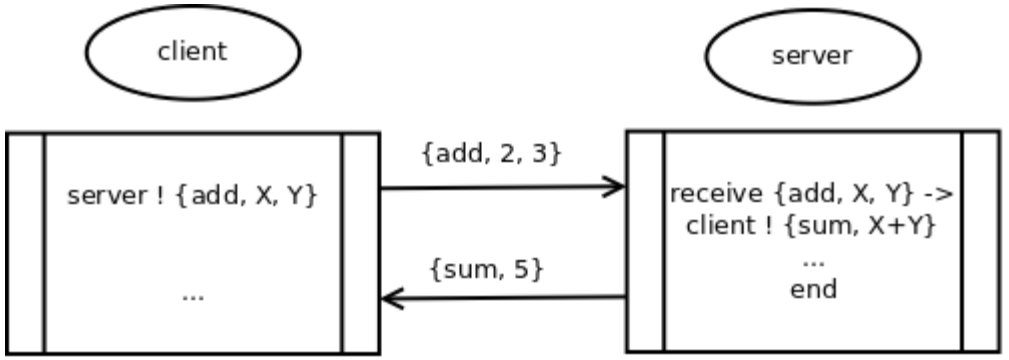
- Erlang (Let it fail)
- Charm ++
- MPI (Checkpoint Restart)

Erlnag

- Функциональный язык и исполнительная система с поддержкой отказоустойчивости. Разработан фирмой Ericsson
- Модель отказоустойчивости — Let it fail

Erlang

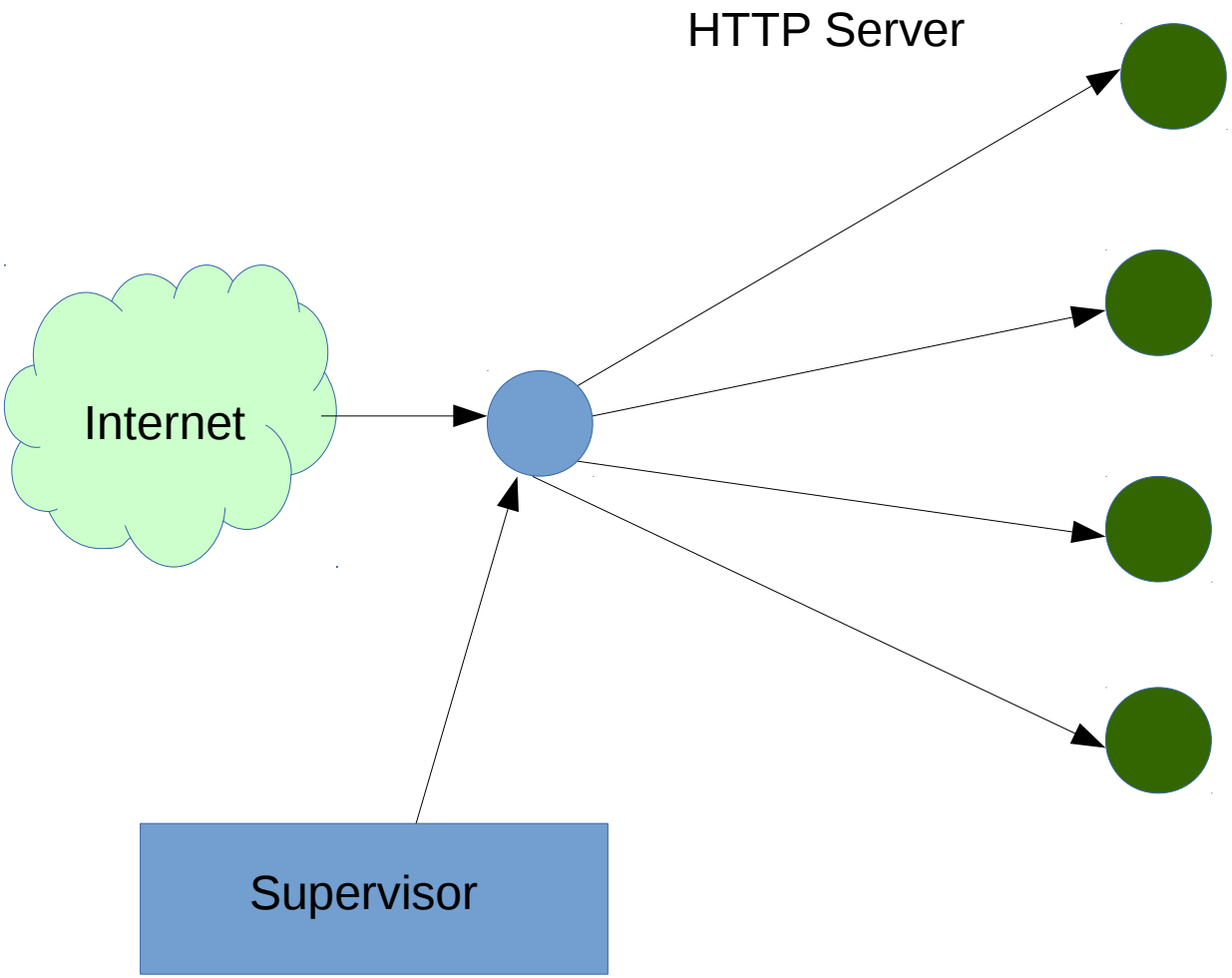
- Erlang использует модель акторов (мини — процессов)
- Каждый процесс может получать и передавать сообщения, порождать новые процессы и принимать локальные решения в зависимости от принятых сообщений



Отказоустойчивость в Erlang

- Let it fail — при завершении процесса остальная система продолжает работать
- Supervisor — распределенный сервис, позволяющий контролировать «важные» процессы и перезапускать их

Отказоустойчивость в Erlang



Минусы Erlang

- Изначально Erlnag предназначен не для параллельных вычислений, а для телекоммуникационных систем
- Идея облегченных процессов предполагает малое потребление ими памяти

Charm ++

- Основной единицей является процесс (объект, чар)
- Процессы взаимодействуют друг с другом с помощью сообщений
- Когда программа вызывает метод объекта, ему посылается сообщение
- Объект может находится как на локальном узле, так и на удаленном
- Система исполнения автоматически распределяет объекты по узлам

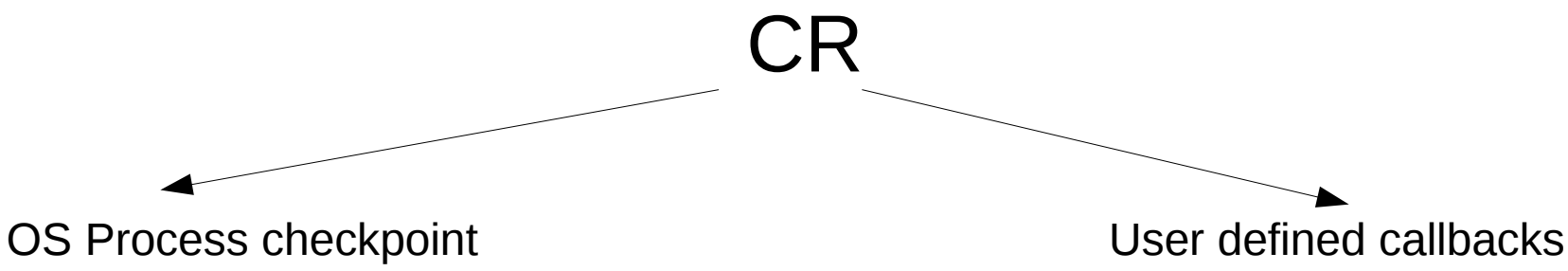
Отказоустойчивость Charm++

- Charm ++ использует алгоритмы Checkpoint \ Restart, при этом состояния сохраняются в нескольких копиях (double-checkpointing)
- Такая система позволяет сохранять не все состояние процесса ОС
- Message Logging

MPI

- Отказоустойчивые реализации MPI используют Checkpoint\Restart, пр этом сохраняется состояние процесса ОС целиком
- Многие реализации используют библиотеку BLCR

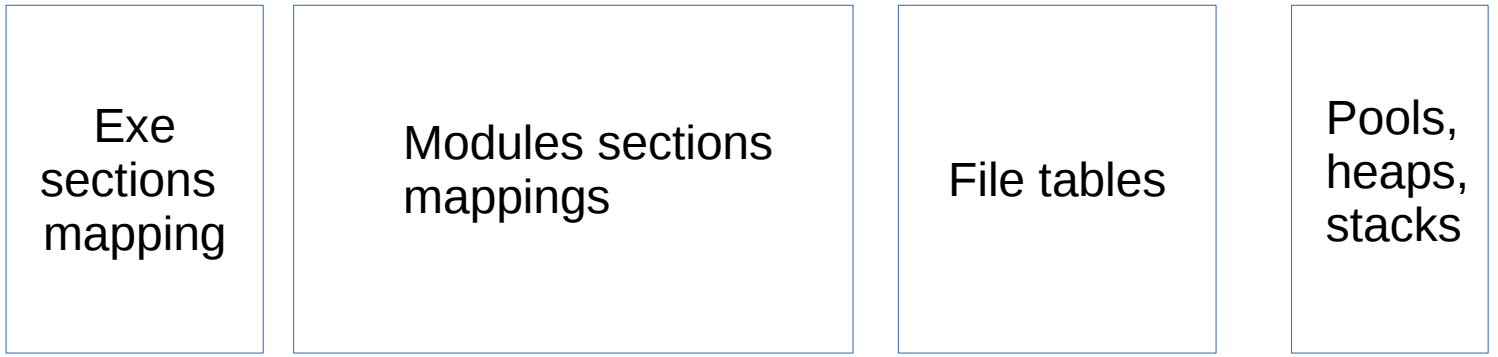
MPI



OS Process checkpoint

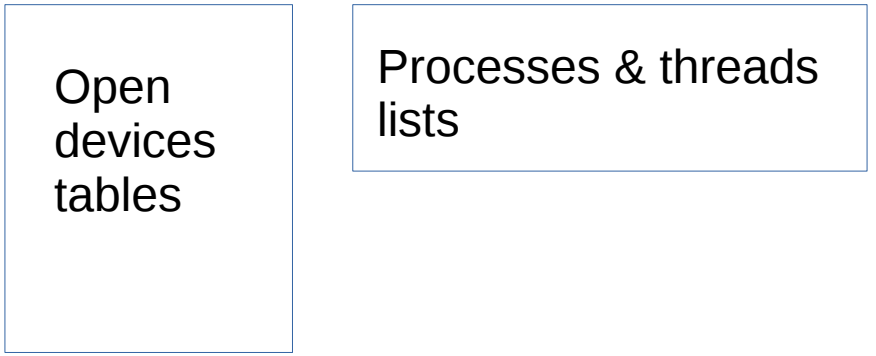
- При полном сохранении состояния процесса сохраняются (при этом необходимо прозрачное сохранение) :
  - PID
  - Child PIDs
  - Opened descriptors
  - VM

OS Process checkpoint

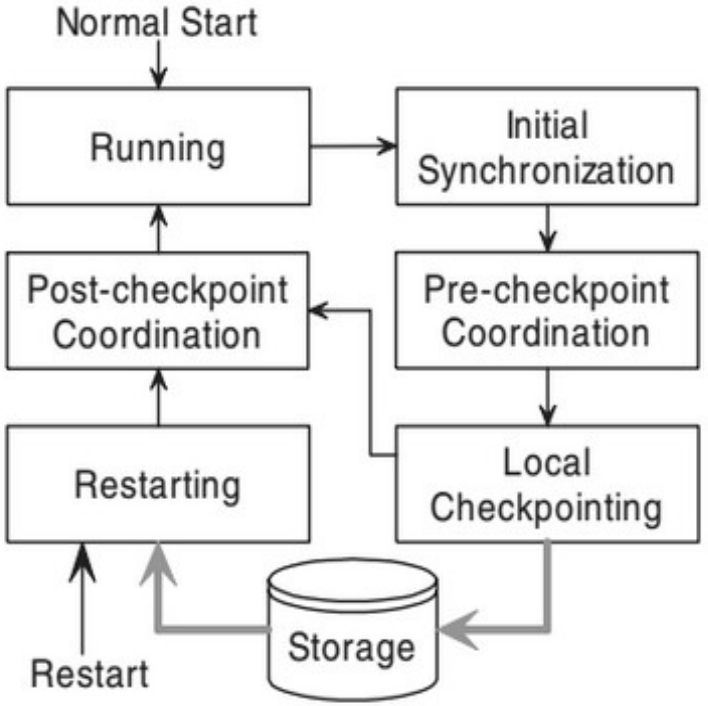


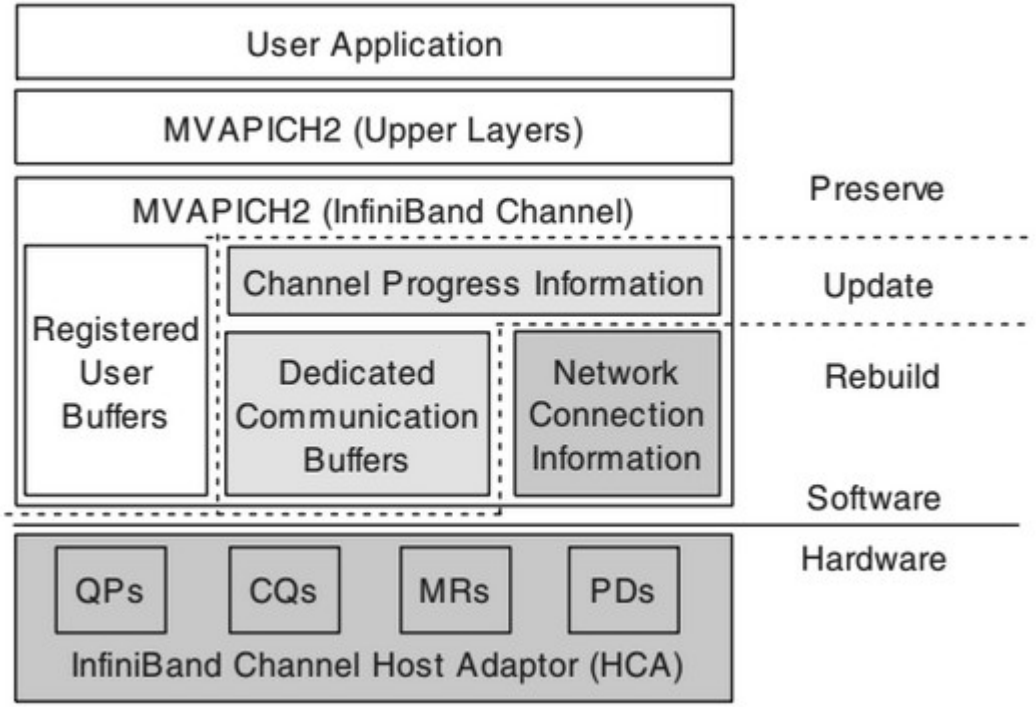
ring - 3 Process VM

ring - 0



Пример алгоритма CR (MVAPICH)





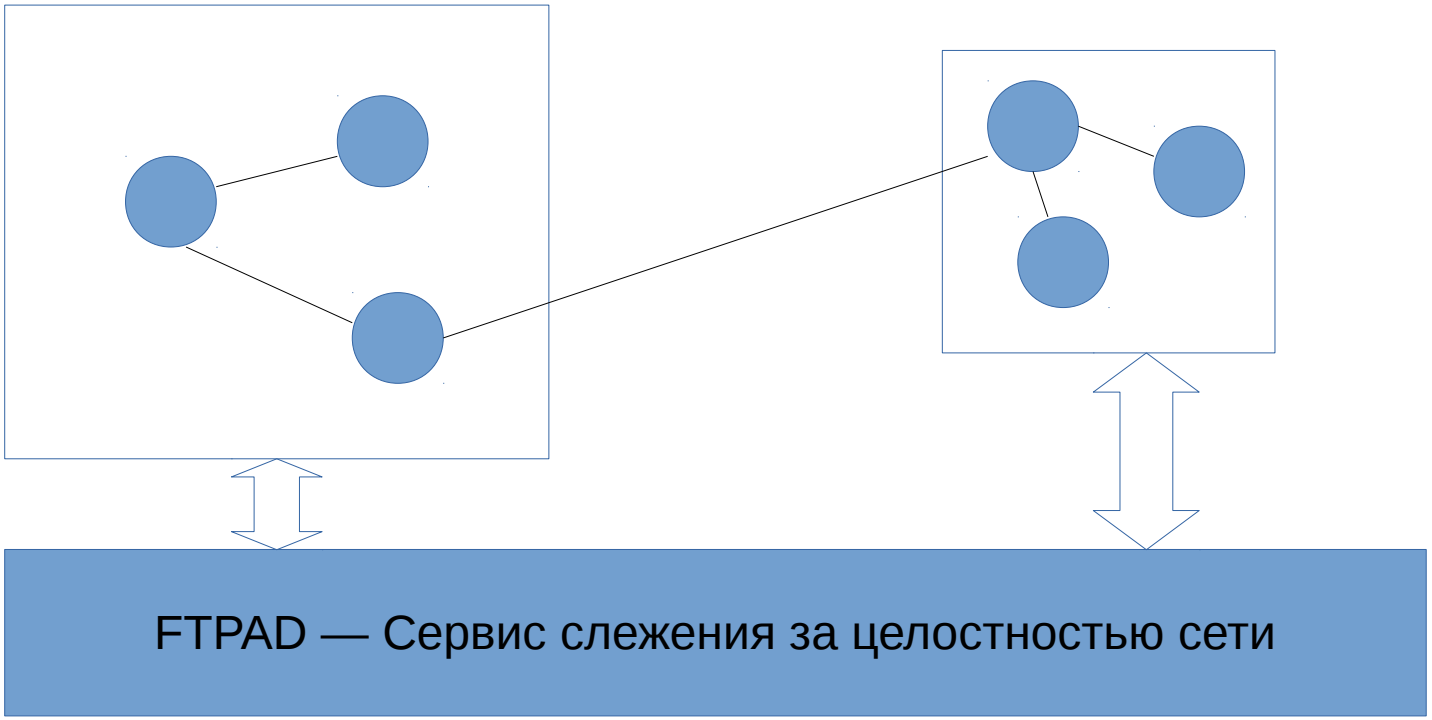
Основные общие проблемы алгоритмов, основанных на CR

- Необходимость синхронизаций
- Избыточность сохраняемых данных
- Сложность получения и восстановления некоторых частей состояния процесса при его полном сохранении

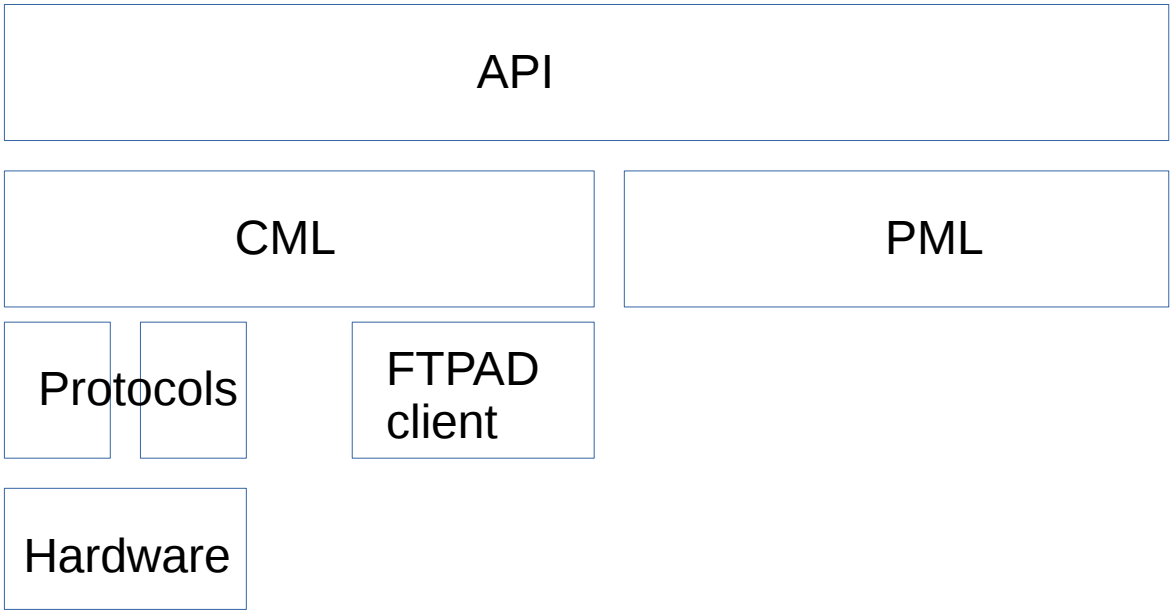
Отказоустойчивость в системе LuNA

FTPA

- Для обеспечения отказоустойчивости в системе LuNA разработан коммуникационный слой FTPA



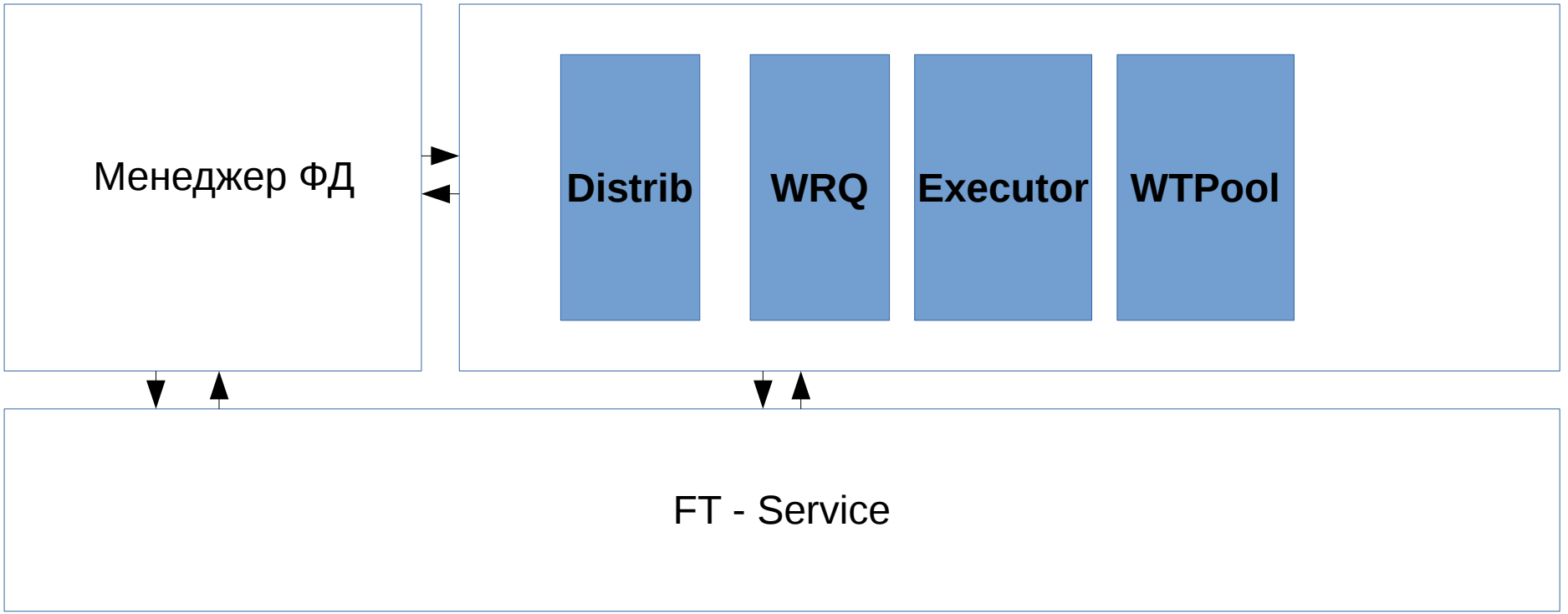
Архитектура FTPA



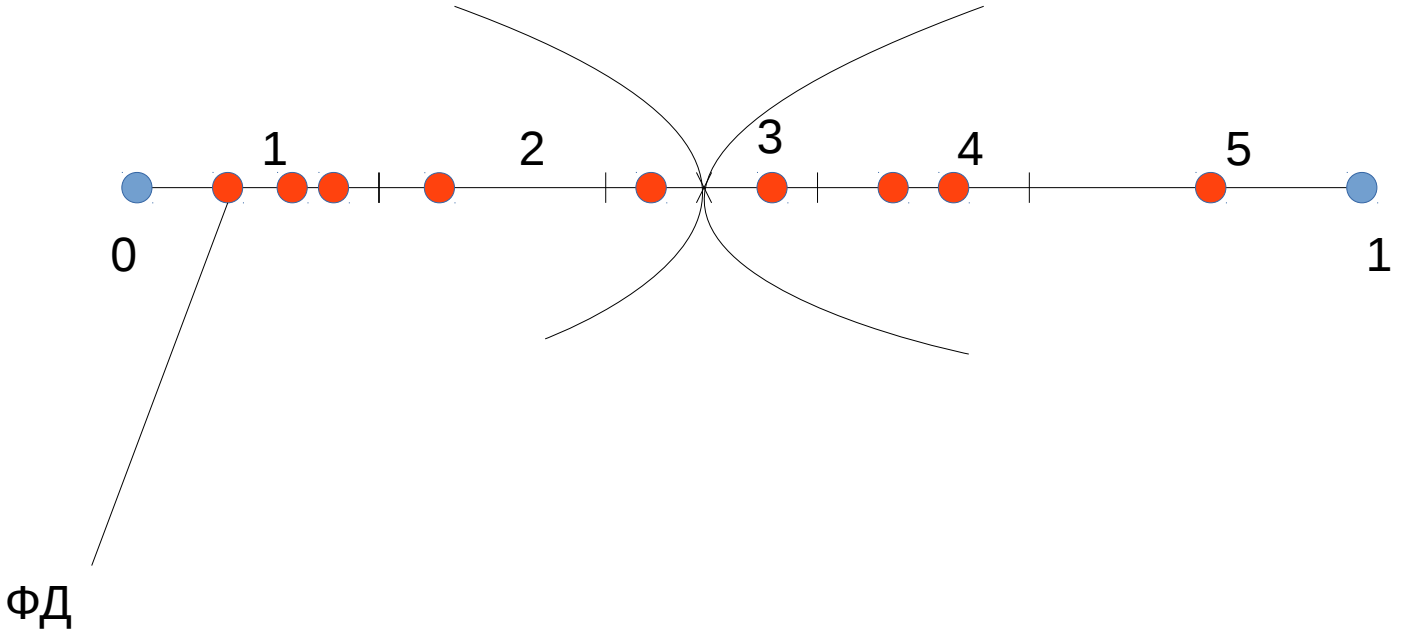
Отказоустойчивость в системе LuNA

- Необходимо создание общего абстрактного класса отказоустойчивого сервиса
- Его будут использовать все компоненты LuNA
- Сервис должен предоставлять интерфейс для сохранения и восстановления состояния компонент системы
- Часть компонент системы с помощью сервиса должна предоставлять необходимые данные для сохранения (дублирования)
- Дополнительный компонент системы должен управлять дублированием ФВ и ФД посредством взаимодействия с остальными компонентами системы с помощью вызова переопределенных методов абстрактного класса

Архитектура



DFM



Общий принцип обеспечения отказоустойчивости

- При отказе одного из узлов суперкомпьютера ФД и ФВ, находившиеся под контролем компонент системы на данном узле должны перейти на работоспособные узлы

Distrib

- На стадии distrib определяется будет ли ФВ исполнен на текущем узле
- Состоянием здесь является множество ФВ, поступающих на исполнение на текущем узле

WRQ

- На стадии WRQ для ФВ ожидается вычисление всех входных ФД
- Сохранять необходимо связки ФД+ФВ

Executor

- На стадии Executor ФВ исполняется
- При исполнении ФВ возможно порождение новые ФВ, они также должны быть продублированы

WTPool

- WTPool исполняет атомарные ФВ и представляет собой очередь задач
- ФВ, находящиеся в очереди также должны быть продублированы

Интерфейс абстрактного отказоустойчивого сервиса

- virtual void lock() = 0;
- virtual void getState(StateItem* items, int count) = 0;
- virtual void addState(StateItem* items, int count) =0 ;
- virtual void unlock() = 0;
-

```
class StateItem : public Serializable
{
int component; // Component id
int type; // 0 - CF 1 - DF
CFTask* cf;
DataFragment* df;
public:
StateItem(CFTask* cf, int component);
StateItem(DataFragment* df, int component);
~StateItem();
};
```

Спасибо за внимание