NOP on multi-core architecture computers

Guilherme Martini

Jean Simão

Robson Linhares

Objetivo do trabalho

- Aprofundar o tema multi-threading para o PON em arquiteturas PC

Definição das etapas principais

1. Estudar maneiras e ferramentas nas quais o multi-threading poderia ser implementado

2. Escolher uma das tecnologias para gerar um caso de estudo comparável com testes anteriores

3. Desenvolver o caso de uso na tecnologia proposta

4. Desenvolver o mesmo caso de uso no método proposto por Belmonte

5. Comparar resultados e evidenciar limitações/vantagens

1 - Tecnologias estudadas

- PThreads
- Thread Pooling
- Erlang
- Haskell
- Node.js
- CSP (msgs do Golang)
- Open CL
- Open MP
- FADALib
- Multi Agent Systems theory (MAS)
- Akka.net

1 - Tecnologias consideradas

- Erlang - Criada na Ericsson para ser distribuída, tolerante a falha e soft-real time. Por já possuir 32 anos, foi optado por usar outras tecnologias mais recentes.
- Node.js - Sistema que sobre uma JVM e interpreta Javascript para criar sistemas multi-agentes. Descartado devido a conhecidos overheads gerados na cadeia de processamento.
- Open MP – API que roda em C++ (e outras) com ferramental para sistemas multithread que compartilham memória. Considerada boa abordagem para futuras melhorias no framework criado por Belmonte.

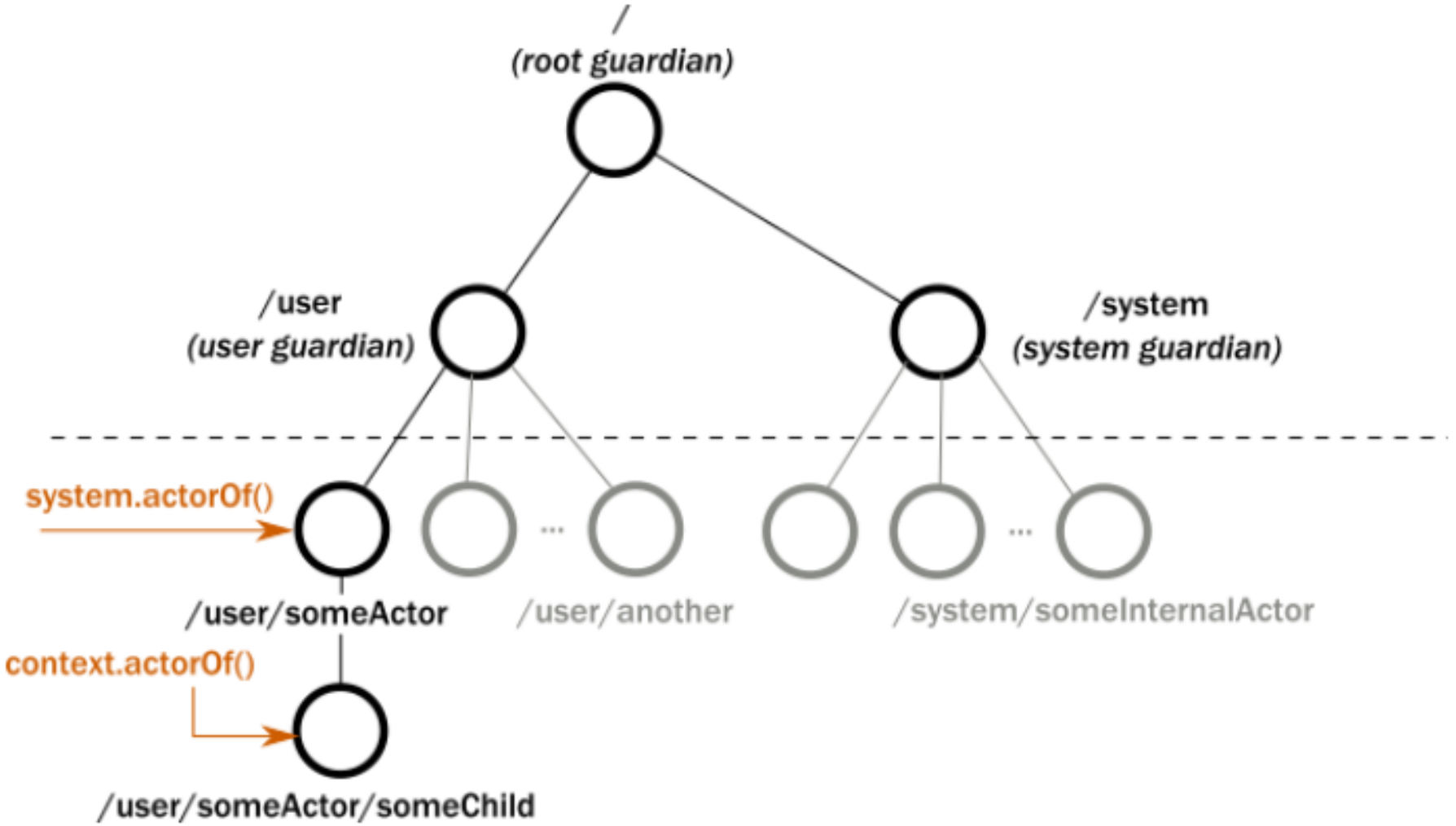
2 - Tecnologia estudada: Akka.net

- É um conjunto de bibliotecas baseadas nos conceitos do Erlang disponível nas linguagens Java, Scala e C#.
- Akka é uma tecnologia que foca na construção de sistema baseado no modelo de atores, o que torna interessante seu uso visto prévias comparações com o PON.
- As ferramentas trazem transparência para implementação distribuída/escalável.
- Há também mecanismos que facilitam a criação de condições de contorno de falha e de criação dinâmica de atores.

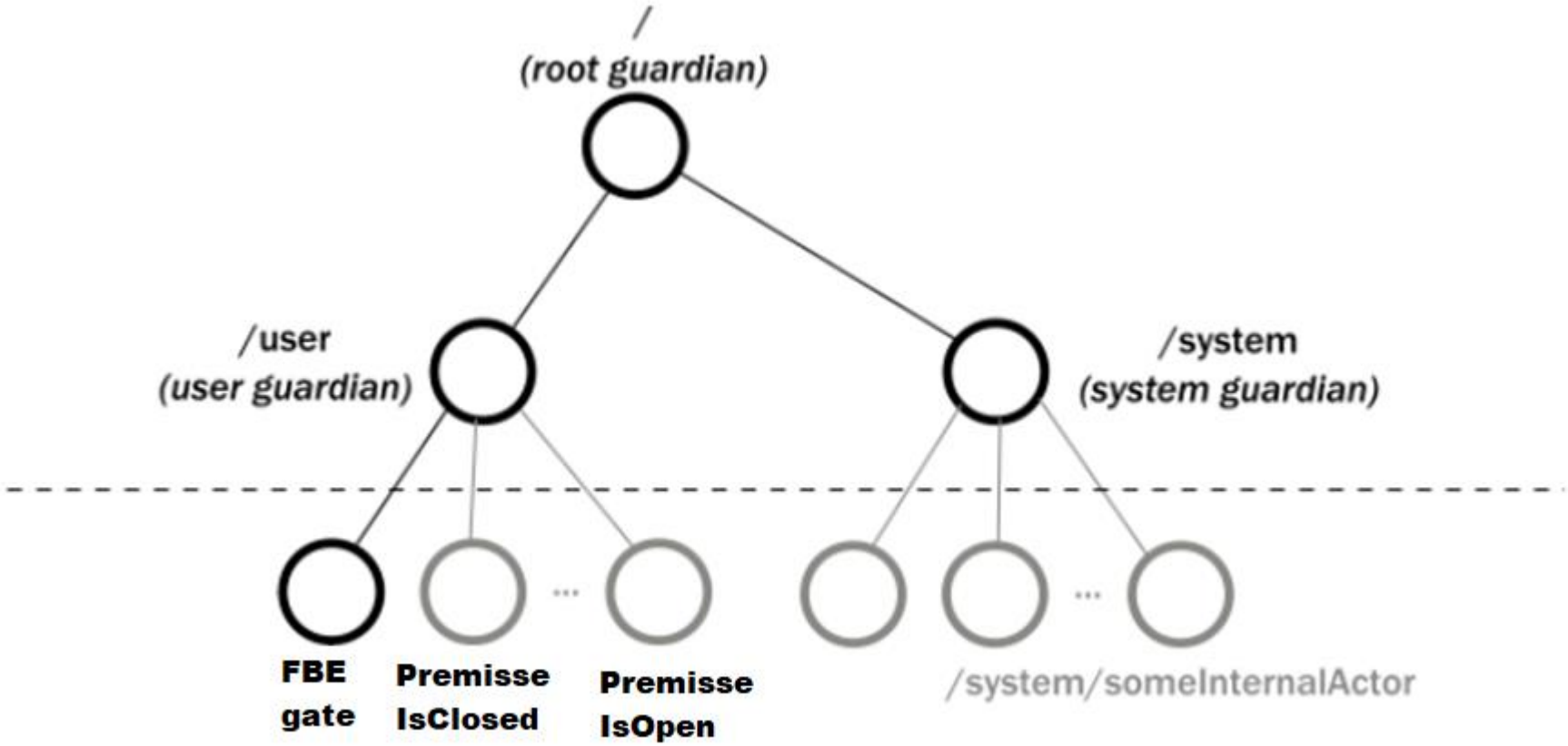
2 - Por que Akka?

- Devido à riqueza de ferramental e facilidade de distribuição, é indagado o quanto este ferramental gera overheads e ineficiência quando comparado à abordagens diretas em C++.

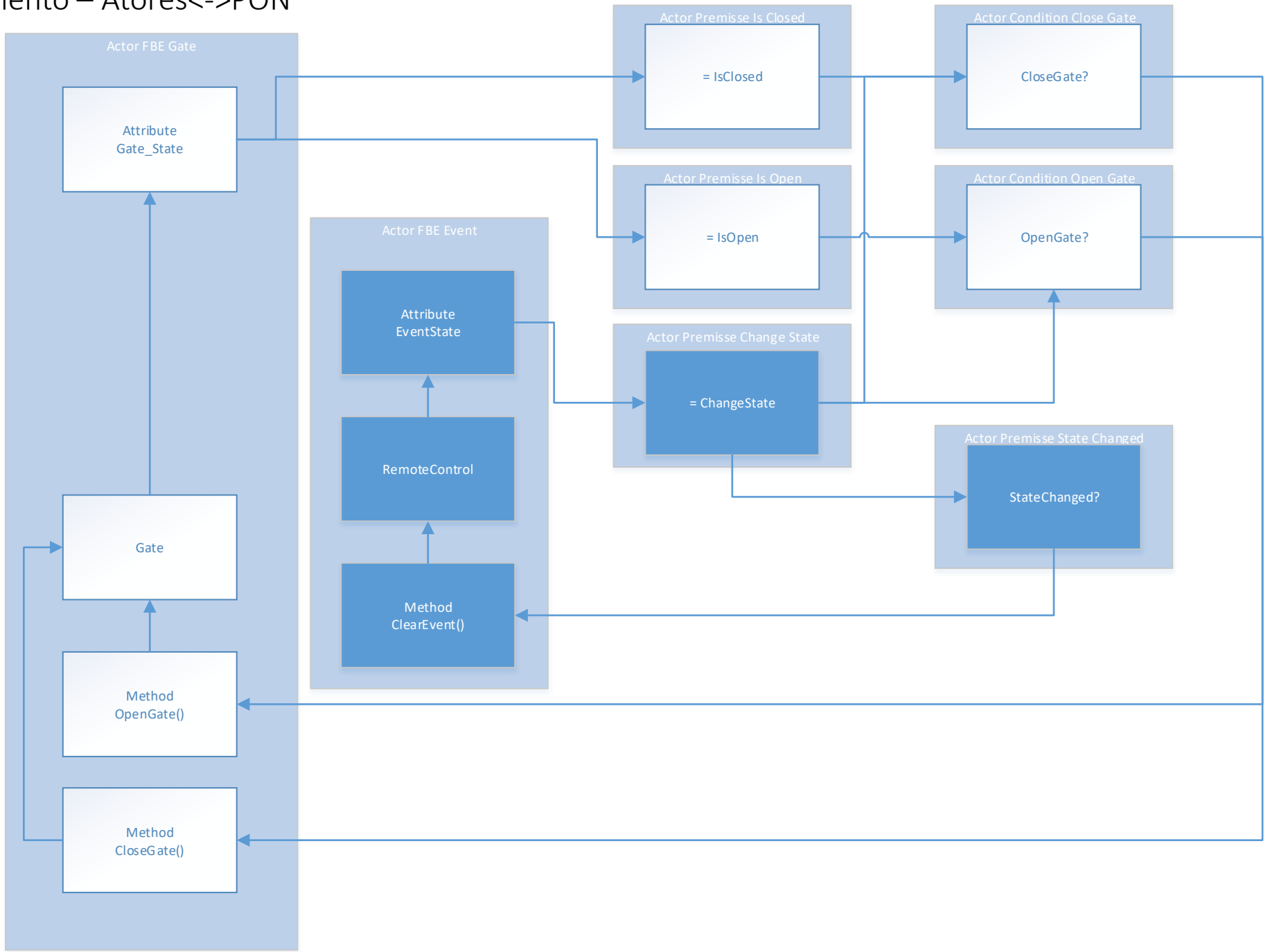
3 – Desenvolvimento – Modelo de atores - Akka



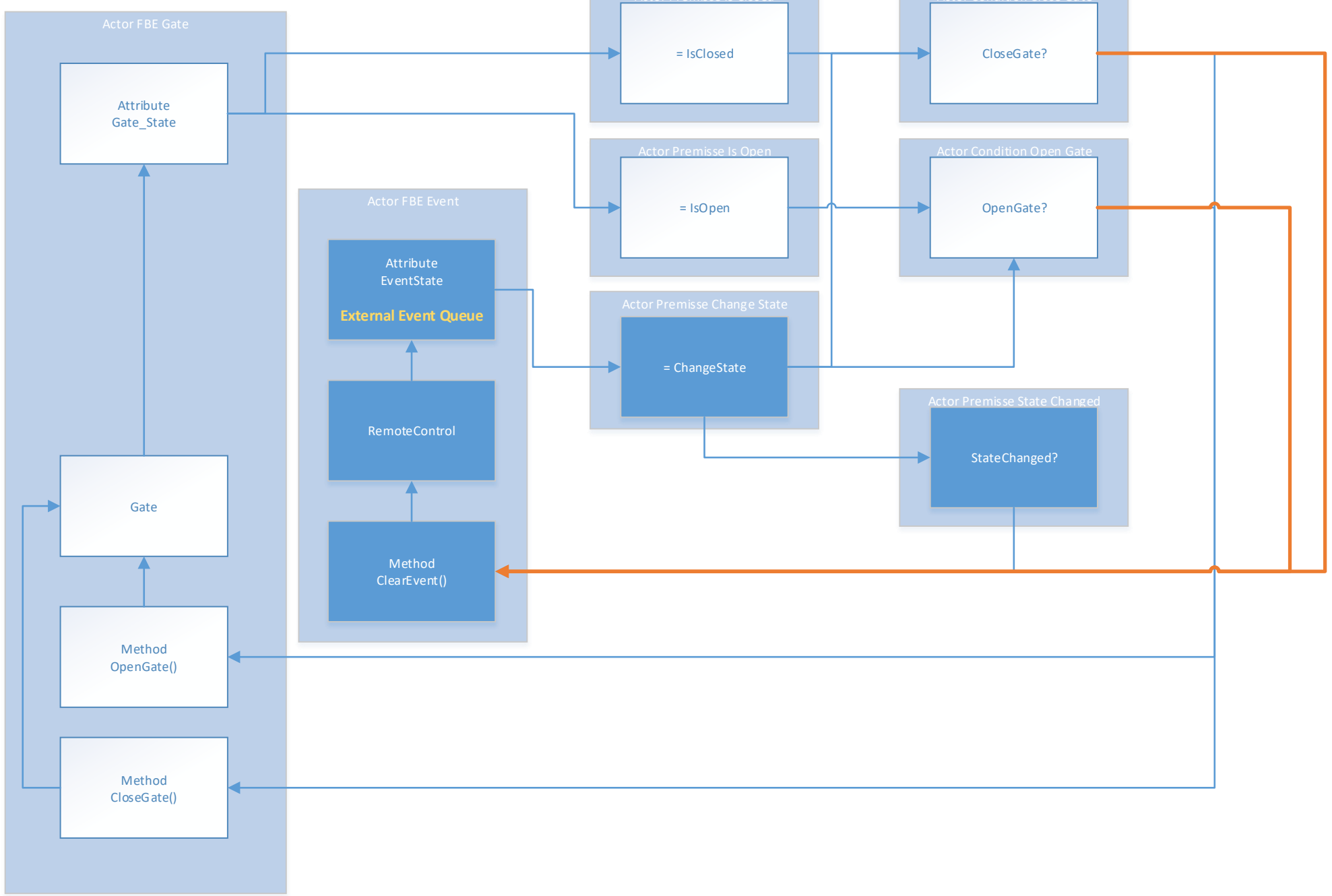
3 – Desenvolvimento – Modelo de atores



3 – Desenvolvimento – Atores<->PON



3 – Desenvolvimento – Atores<->PON



3 – Como um ator simples é implementado

```
310     class ConditionStateChanged : UntypedActor
311     {
312         private IActorRef _FBERemoteControlRef;
313
314
315         private void SetFBERemoteControl(IActorRef FBERemoteControl)
316         {
317             _FBERemoteControlRef = FBERemoteControl;
318         }
319
320         protected override void OnReceive(object message)
321         {
322             if (message is ActorReference)
323             {
324                 var temp = message as ActorReference;
325                 if (temp.ActorRefID == ActorRefType.FBERemoteControlRef) SetFBERemoteControl(temp.Ref);
326
327                 Sender.Tell(true);
328             }
329             else switch (message)
330             {
331                 case ConditionAction.SendFalse:
332                     if (!_FBERemoteControlRef.IsNobody()) _FBERemoteControlRef.Tell(RemoteControlState.Off);
333                     Crc32_many_times();
334                     break;
335             }
336         }
337     }
338 }
```

3 – Como os atores são declarados

```
//Build up all actors
IActorRef FBEGateActor              = NOPActorSystem.ActorOf(Props.Create(() => new FBEGate()),              "FBEGateActor");
IActorRef FBERemoteControlActor     = NOPActorSystem.ActorOf(Props.Create(() => new FBERemoteControl()),     "FBERemoteControlActor");
IActorRef PremisseIsClosedActor     = NOPActorSystem.ActorOf(Props.Create(() => new PremisseIsClosed()),     "PremisseIsClosedActor");
IActorRef PremisseIsOpenActor       = NOPActorSystem.ActorOf(Props.Create(() => new PremisseIsOpen()),       "PremisseIsOpenActor");
IActorRef PremisseChangeStateActor  = NOPActorSystem.ActorOf(Props.Create(() => new PremisseChangeState()),  "PremisseChangeStateActor");
IActorRef ConditionCloseGateActor   = NOPActorSystem.ActorOf(Props.Create(() => new ConditionCloseGate()),   "ConditionCloseGateActor");
IActorRef ConditionOpenGateActor    = NOPActorSystem.ActorOf(Props.Create(() => new ConditionOpenGate()),    "ConditionOpenGateActor");
IActorRef ConditionStateChangedActor = NOPActorSystem.ActorOf(Props.Create(() => new ConditionStateChanged()), "ConditionStateChangedActor");
```

3 – Como as notificações são endereçadas

```
//Set all actor references
var FBEGateActorTask1              = FBEGateActor.Ask(new ActorReference(ActorRefType.PremisseIsClosedRef, PremisseIsClosedActor));
var FBEGateActorTask2              = FBEGateActor.Ask(new ActorReference(ActorRefType.PremisseIsOpenRef, PremisseIsOpenActor));
var FBERemoteControlActorTask1     = FBERemoteControlActor.Ask(new ActorReference(ActorRefType.PremisseChangeStateRef, PremisseChangeStateActor));
var PremisseIsClosedActorTask1     = PremisseIsClosedActor.Ask(new ActorReference(ActorRefType.ConditionCloseGateRef, ConditionCloseGateActor));
var PremisseIsOpenActorTask1       = PremisseIsOpenActor.Ask(new ActorReference(ActorRefType.ConditionOpenGateRef, ConditionOpenGateActor));
var PremisseChangeStateActorTask1  = PremisseChangeStateActor.Ask(new ActorReference(ActorRefType.ConditionCloseGateRef, ConditionCloseGateActor));
var PremisseChangeStateActorTask2  = PremisseChangeStateActor.Ask(new ActorReference(ActorRefType.ConditionOpenGateRef, ConditionOpenGateActor));
var PremisseChangeStateActorTask3  = PremisseChangeStateActor.Ask(new ActorReference(ActorRefType.ConditionStateChangedRef, ConditionStateChangedActor));
var ConditionCloseGateActorTask1   = ConditionCloseGateActor.Ask(new ActorReference(ActorRefType.FBEGateRef, FBEGateActor));
var ConditionCloseGateActorTask2   = ConditionCloseGateActor.Ask(new ActorReference(ActorRefType.FBERemoteControlRef, FBERemoteControlActor));
var ConditionOpenGateActorTask1    = ConditionOpenGateActor.Ask(new ActorReference(ActorRefType.FBEGateRef, FBEGateActor));
var ConditionOpenGateActorTask2    = ConditionOpenGateActor.Ask(new ActorReference(ActorRefType.FBERemoteControlRef, FBERemoteControlActor));
var ConditionStateChangedActorTask1 = ConditionStateChangedActor.Ask(new ActorReference(ActorRefType.FBERemoteControlRef, FBERemoteControlActor));
```

3 - Como iniciar a aplicação

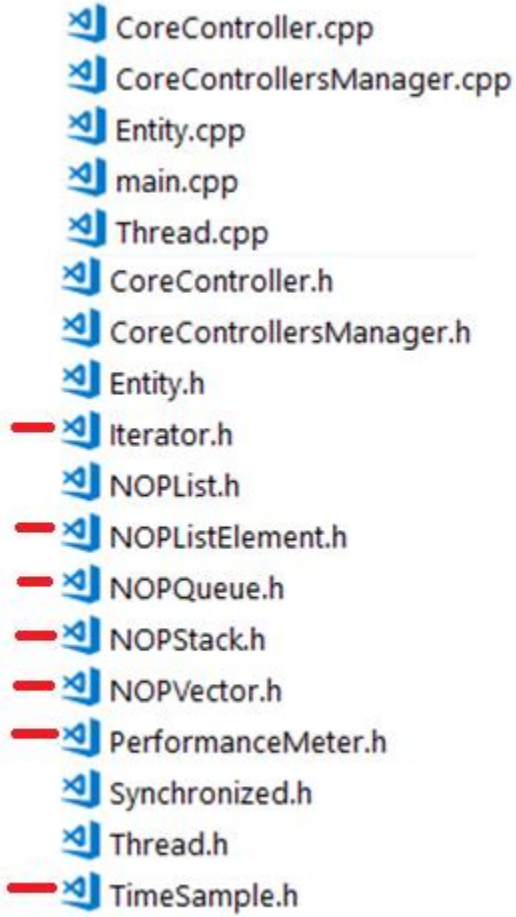
```
//ActionTriggers
int i = 4;
while((i--) > 0) FBERemoteControlActor.Tell(RemoteControlState.ButtonPress);

//Blocks the main thread from exiting until the actor system is shut down
NOPActorSystem.WhenTerminated.Wait();
```

3 – Função de carga utilizada para testes

```
// calculate a checksum on a buffer -- start address = p, length = bytelength
unsigned int crc32_byte(unsigned char *p, unsigned int bytelength)
{
    unsigned int crc = 0xffffffff;
    while (bytelength-- !=0) crc = poly8_lookup[((unsigned char) crc ^ *(p++))] ^ (crc >> 8);
    // return (~crc); also works
    return (crc ^ 0xffffffff);
}

void crc32_many_times(void)
{
for(int i=0; i<CRC32_N_LOOPS; i++) crc32_byte(buffer, 50);
}
```

```
const unsigned int poly8_lookup[256] =
{
    0, 0x77073096, 0xEE0E612C, 0x990951BA,
    0x076DC419, 0x706AF48F, 0xE963A535, 0x9E6495A3,
    0x0EDB8832, 0x79DCB8A4, 0xE0D5E91E, 0x97D2D988,
    0x09B64C2B, 0x7EB17CBD, 0xE7B82D07, 0x90BF1D91,
    0x1DB71064, 0x6AB020F2, 0xF3B97148, 0x84BE41DE,
    0x1ADAD47D, 0x6DDDE4EB, 0xF4D4B551, 0x83D385C7,
    0x136C9856, 0x646BA8C0, 0xFD62F97A, 0x8A65C9EC,
    0x14015C4F, 0x63066CD9, 0xFA0F3D63, 0x8D080DF5,
    0x3B6E20C8, 0x4C69105E, 0xD56041E4, 0xA2677172,
```

4 – Desenvolvimento em C++ - Framework Multithread
Estrutura do projeto resumido



4 – Desenvolvimento em C++ - Framework Multithread
Criação e amarração de entidades

```
CoreControllersManager::createCoreControllers(8);

//Gate notification cycle
Entity *attribute1 = new Entity("gate_state");
attribute1->setCore(0);

Entity *premise1 = new Entity("premise_is_closed");
premise1->setCore(1);
attribute1->registerEntity(premise1);

Entity *premise2 = new Entity("premise_is_open");
premise2->setCore(2);
attribute1->registerEntity(premise2);

Entity *condition1 = new Entity("condition_close_gate");
condition1->setCore(3);
premise1->registerEntity(condition1);
```

```
// Start application...
int i = 1;
while(i <= 4)
{
    //std::cout << "Button Press: " << i << std::endl;
    attribute2->onNotification();

    while(method1->exec_count < (2*i) ||
          method2->exec_count < (2*i) ||
          attribute1->exec_count < i  ||
          premise1->exec_count   < i  ||
          premise2->exec_count   < i  ||
          condition1->exec_count < i  ||
          condition2->exec_count < i  ||
          attribute2->exec_count < i );

    i++;
}
```

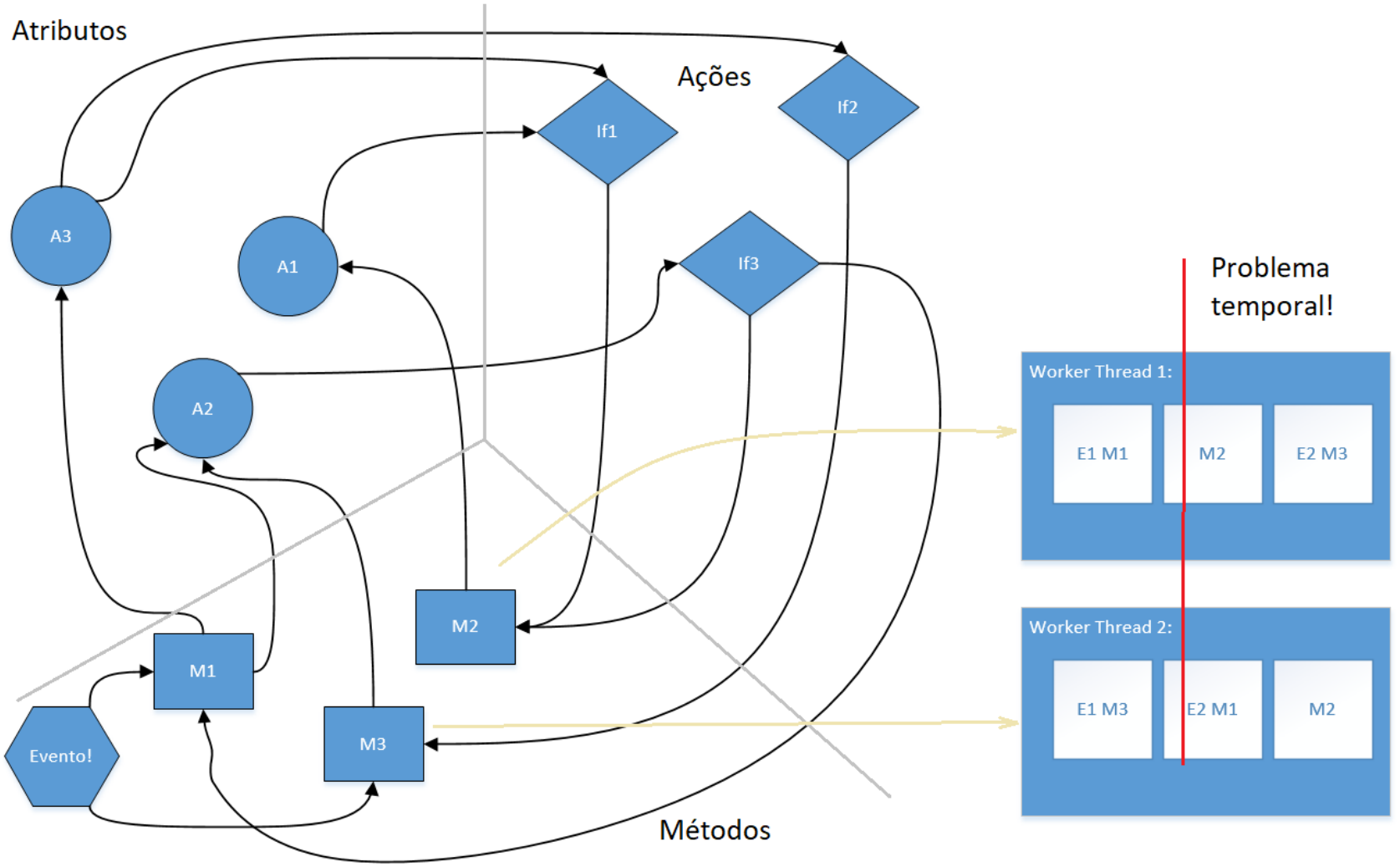
4 – Desenvolvimento em C++ - Framework Multithread
Entidades <-> Atores

```
void Entity::setCore(short id) {
    coreId = id;
}

void Entity::registerEntity(Entity *entity) {
    entities.addLastElement(entity);
}

void Entity::onNotification() {
    CoreControllersManager::registerNotifier(coreId, this);
}

void Entity::execute() {

    //std::cout << "Executing " << name << " ["<< coreId << "]" << std::endl;

    entities.initIterator();

    while (entities.hasNext()) {
        entities.next()->onNotification();
    }

    //Method function calls

    exec_count++;

}
```

```
void CoreControllersManager::registerNotifier(short coreId, Entity *entity) {
    cores.getElement(coreId)->addNotifier(entity);
}
```

```
void CoreController::addNotifier(Entity *entity) {

    notifiers.push(entity);

    if (!running)
        this->start();
}

void CoreController::run() {

    running = true;

    while (!notifiers.isEmpty())
        notifiers.pop()->execute();

    running = false;
}
```

4 – Desenvolvimento em C++ - Framework Multithread
Fluxo de notificações

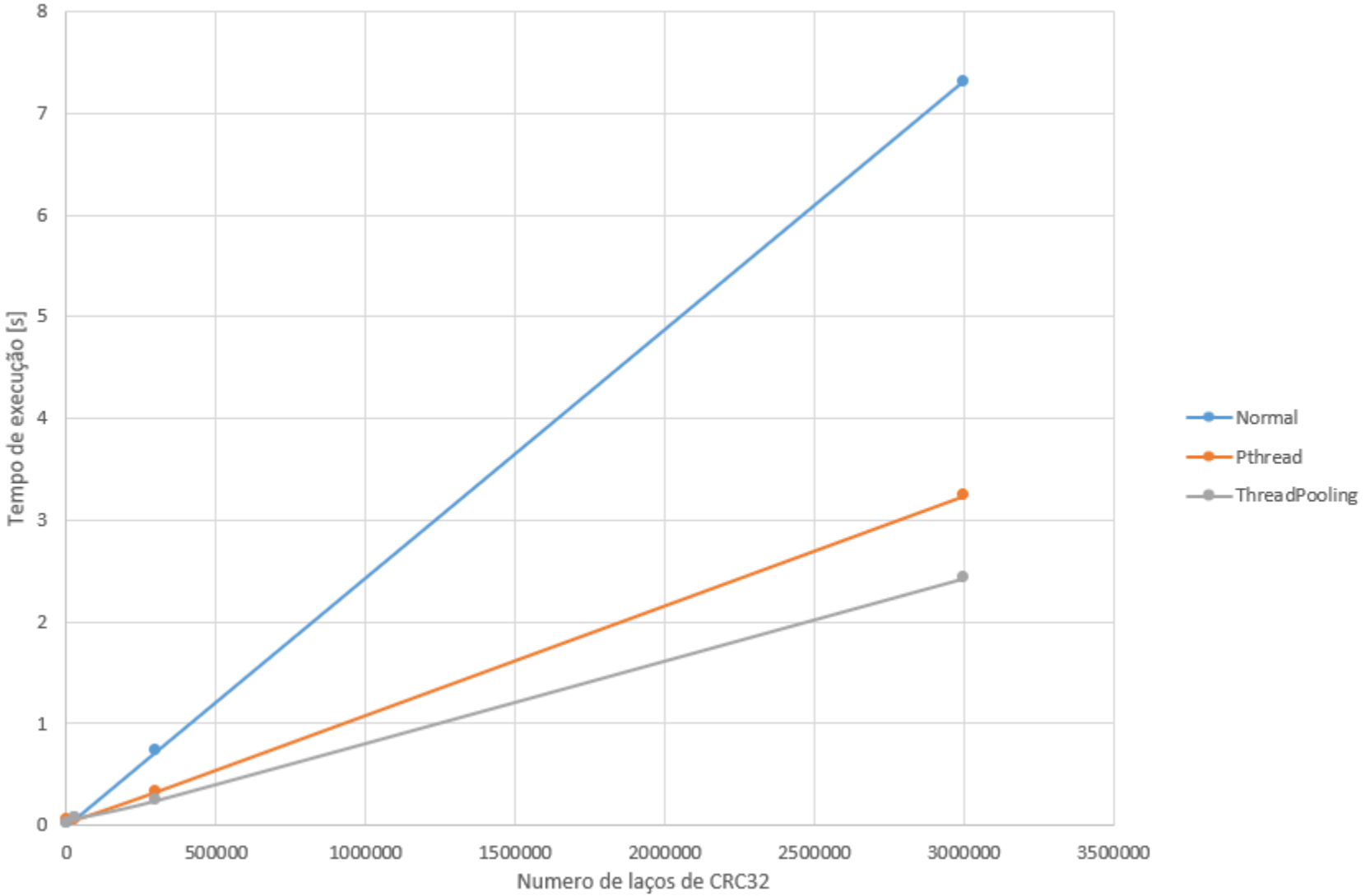


5 – Resultados prévios

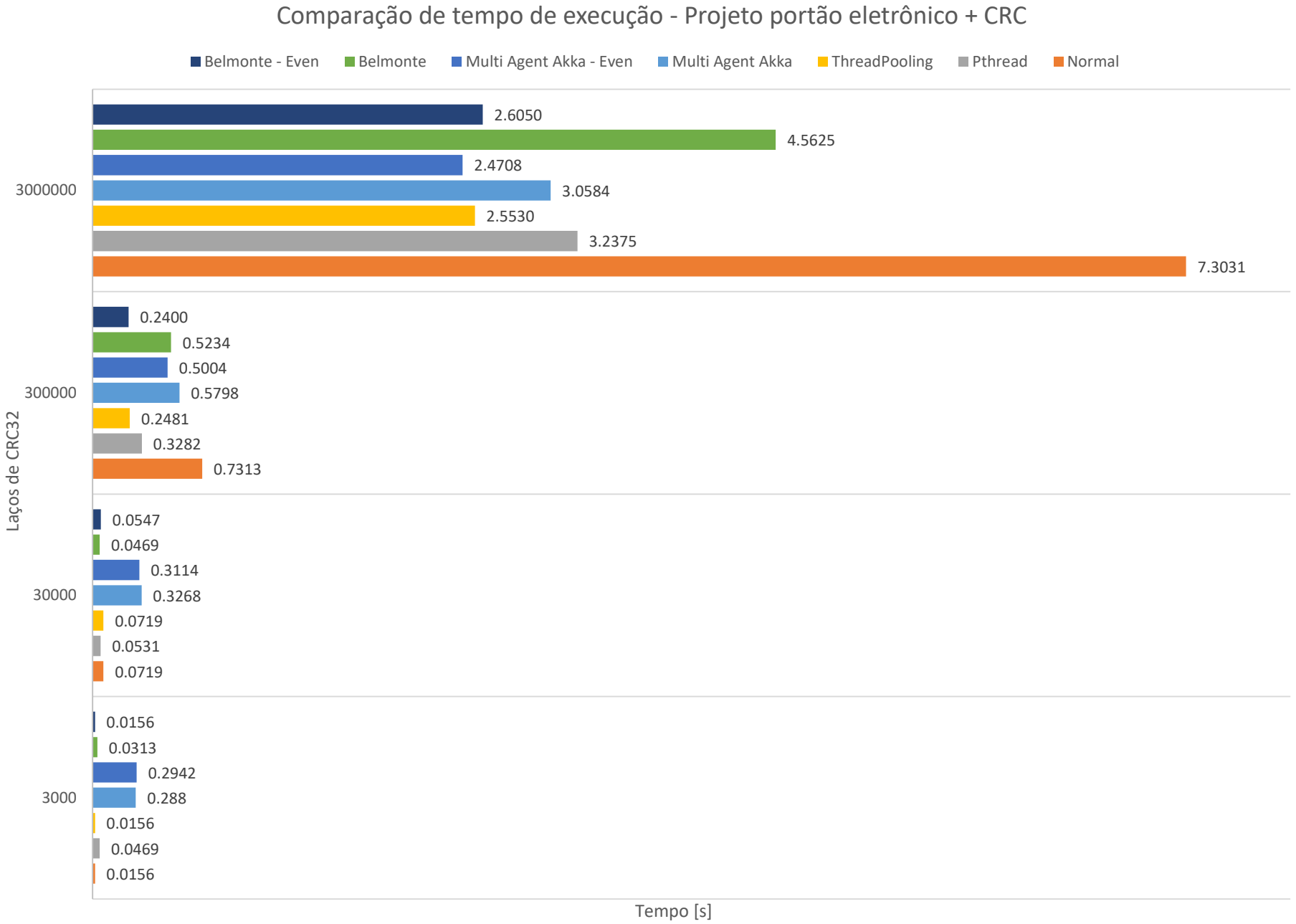
Normal 3000	Normal 30000	Normal 300000	Normal 3000000	
0.0156	0.0625	0.7344	7.3438	
0.0156	0.0625	0.7188	7.2969	
0.0156	0.0781	0.7344	7.2813	
0.0156	0.0781	0.7344	7.2969	
0.0156	0.0781	0.7344	7.2969	
0.015625	**0.071875**	**0.73125**	**7.303128**	**Media**
0	**0.008558165**	**0.006987712**	**0.023695497**	**DesvPad**

PThread 300(	PThread 30000	PThread 300000	PThread 3000000	
0.0469	0.0625	0.3282	3.2813	
0.0469	0.0469	0.3282	3.1719	
0.0469	0.0625	0.3282	3.1719	
0.0469	0.0469	0.3282	3.2813	
0.0469	0.0469	0.3282	3.2813	
0.046875	**0.053125**	**0.32815**	**3.237502**	**Media**
0	**0.008558165**	**0**	**0.059904416**	**DesvPad**

TPooling 300	TPooling 30000	TPooling 30000	TPooling 3000000	
0.0156	0.0625	0.2656	2.5781	
0.0156	0.0781	0.2688	2.5313	
0.0156	0.0781	0.2344	2.6094	
0.0156	0.0625	0.2688	1.8750	
0.0156	0.0781	0.2031	2.5313	
0.015625	**0.071875**	**0.248125**	**2.425**	**Media**
0	**0.008558165**	**0.02903056**	**0.309240943**	**DesvPad**



5 - Resultados



5 - Resultados

- Diferença não significativa em tempo de execução entre abordagens do Belmonte e Akka quando há carga computacional representativa no modo “even”.
- Akka é favorável para expansão em sistemas distribuídos/clusters.
- Akka já possui arquitetura para tolerância de falha e criação dinâmica de atores.
- Já existe a possibilidade para criação de FSM e passagem de objetos abstratos como notificação no Akka.
- Framework do Belmonte ainda com falta de aplicações finalizada. Falta de tratamentos de sincronia entre entidades e quebra do paradigma de modelo de atores ainda não terminado, mas pode ser otimizada ainda mais para aplicações multi-thread.
- Open MP poderia ser usado no framework do Belmonte para mais ganhos.

5 - Links relevantes:

- http://www.labri.fr/perso/barthou/ps/main_fadalib.pdf
- http://www.eventiotic.com/eventiotic/files/Papers/URL/540b95d8-b703-47c2-96b8-ad5131ab67ed.pdf
- https://getakka.net/
- https://www.openmp.org/

Obrigado!