Tracing best practices

Marcin Mrowiński

Senior Software Engineer

brightONE

marcin.mrowinski@brightone.pl

Tracing best practices

- Agenda:
  - Trace
  - Good & wrong traces
  - Examples
  - Summary
  - Additional hints
  - Advanced example
  - Questions?

Trace

Traces in real life

- Worst traces
  -

Traces in real life

- Worst traces
  -
- Wrong trace:
  - Hello everyone, let’s start training

Traces in real life

- Worst traces
  -
- Wrong trace:
  - Hello everyone, let’s start training
- Better trace:
  - Hello, I’m Marcin Mrowiński, I’m Senior Software Engineer and in BrightONE. Trainng is about how to properly use traces in code

Traces in real life

- Worst traces
  -
- Wrong trace:
  - Hello everyone, let’s start training
- Better trace:
  - Hello, I’m Marcin Mrowiński, I’m Senior Software Engineer and in BrightONE. Trainng is about how to properly use traces in code
- Best trace:
  - Hi, I’m Marcin Mrowiński. P4 in BrightONE. Training about presentation topic

Traces example

```
int main(int argc, char* argv[])
{
    printf("Example calculations\n");

    int counter = 10;
    printf("Start calculation for %d\n",counter);

    int sum = 0;
    for(int i=0; i<counter; ++i)
    {
        sum = sum + 2;
        printf("Iteration %d/%d, temp sum = %d\n",i,counter,sum);
    }

    printf("Result = %d\n",sum);

    return 0;
}
```

```
Example calculations
Start calculation for 10
Iteration 0/10, temp sum = 2
Iteration 1/10, temp sum = 4
Iteration 2/10, temp sum = 6
Iteration 3/10, temp sum = 8
Iteration 4/10, temp sum = 10
Iteration 5/10, temp sum = 12
Iteration 6/10, temp sum = 14
Iteration 7/10, temp sum = 16
Iteration 8/10, temp sum = 18
Iteration 9/10, temp sum = 20
Calc for 10 = 20
```

Traces example – Android

```
05-29 14:19:18.399 1335-1335/? I/MotoNetwCtrlr: onReceive: Completed intent=Intent { act=android.net.wifi.RSSI_CHANGED flg=0x400001
05-29 14:19:18.417 1335-1335/? I/MotoNetwCtrlr: onReceive: Received intent=Intent { act=android.net.wifi.RSSI_CHANGED flg=0x4000010
05-29 14:19:18.421 1335-1335/? I/MotoNetwCtrlr.MotoWifiSignalCtrlr: handleBroadcast: Entered: Intent=Intent { act=android.net.wifi.
05-29 14:19:18.421 1335-1335/? I/MotoNetwCtrlr.MotorolaWifiSignalController: isDirty: returns false
05-29 14:19:18.421 1335-1335/? I/MotoNetwCtrlr.MotoWifiSignalCtrlr: handleBroadcast: Completed
05-29 14:19:18.421 1335-1335/? I/MotoNetwCtrlr: onReceive: Completed intent=Intent { act=android.net.wifi.RSSI_CHANGED flg=0x400001
05-29 14:19:18.608 1335-1335/? I/MotoNetwCtrlr.MotoNetwCtrlrImpl.MotorolaMobileSignalController( 1 ): MotorolaMobilePhoneStateListe
05-29 14:19:18.608 1335-1335/? I/MotoNetwCtrlr.MotoNetwCtrlrImpl.MotorolaMobileSignalController( 1 ): isDirty: returns true
05-29 14:19:18.608 1335-1335/? I/MotoNetwCtrlr.MotoNetwCtrlrImpl.MotorolaMobileSignalController( 1 ): notifyListeners: calling QS [
05-29 14:19:18.609 1335-1335/? I/MotoNetwCtrlr.MotoNetwCtrlrImpl.MotorolaMobileSignalController( 1 ): notifyListeners: calling QS [
05-29 14:19:18.617 1335-1335/? I/MotoNetwCtrlr.MotoNetwCtrlrImpl.MotorolaMobileSignalController( 1 ): notifyListeners: calling SB [
05-29 14:19:18.618 1335-1335/? I/MotoNetwCtrlr.MotoNetwCtrlrImpl.MotorolaMobileSignalController( 1 ): notifyListeners: calling SB [
05-29 14:19:18.620 1335-1335/? I/MotoNetwCtrlr.MotoNetwCtrlrImpl.MotorolaMobileSignalController( 1 ): notifyListeners: calling SB [
05-29 14:19:18.620 1335-1335/? E/MotoNetwCtrlr: getDataController: No data sim selected
05-29 14:19:18.620 1335-1335/? I/MotoNetwCtrlr.MotoNetwCtrlrImpl.MotorolaMobileSignalController( 1 ): MotorolaMobilePhoneStateListe
05-29 14:19:19.693 834-1276/? E/WifiStateMachine: WifiStateMachine CMD_START_SCAN source -2 txSuccessRate=0,00 rxSuccessRate=0,00 t
05-29 14:19:19.694 834-1276/? E/WifiStateMachine: startDelayedScan send -> 6436 milli 20000
05-29 14:19:19.694 834-1276/? E/WifiStateMachine: WifiStateMachine CMD_START_SCAN with age=20005 interval=30000 maxinterval=300000
05-29 14:19:19.694 834-1276/? E/WifiStateMachine: WifiStateMachine CMD_START_SCAN full=false
05-29 14:19:19.694 834-1276/? E/WifiStateMachine: WifiStateMachine starting scan for "BrightOneGuest"WPA_PSK with 2412,2462,2437
05-29 14:19:19.703 834-1276/? E/WifiStateMachine: [1 496 060 359 702 ms] noteScanstart no scan source uid -2
05-29 14:19:19.870 483-530/? D/TCMD: NL - Read 56 bytes from update socket.
05-29 14:19:19.870 483-530/? D/TCMD: NL - message type is RTM_NEWLINK
05-29 14:19:19.870 483-530/? D/TCMD: Listening for incoming client connection request
05-29 14:19:19.883 834-1276/? E/WifiStateMachine: [1 496 060 359 883 ms] noteScanEnd no scan source onTime=0
05-29 14:19:19.888 834-1276/? E/WifiStateMachine: wifi setScanResults statecom.android.server.wifi.WifiStateMachine$ConnectedState@
05-29 14:19:21.424 1335-1335/? I/MotoNetwCtrlr: onReceive: Received intent=Intent { act=android.net.wifi.RSSI_CHANGED flg=0x4000010
05-29 14:19:21.425 1335-1335/? I/MotoNetwCtrlr.MotoWifiSignalCtrlr: handleBroadcast: Entered: Intent=Intent { act=android.net.wifi.
05-29 14:19:21.425 1335-1335/? I/MotoNetwCtrlr.MotorolaWifiSignalController: isDirty: returns true
```

Traces purpose

„Any non-trivial program contains at least one bug”

- Anonymous

You need **TRACES** to find it!

Traces purpose

- Trace program execution
- Possible to check application state
- Easy code analysis

- Traces characteristic:
  - Provided by testers & developers
  - Contains low level information
  - Generally noisy
  - No limitation to output format
  - Does not need localization
  - Can be added almost anywhere

Trace story

- Other developers should be able to understand teammates traces.
- Testers should be able to understand traces

- Learn the traces that are vital to test scenarios!
  - Input (screen, button)
  - Audio situation
  - Device detection
  - HMI screen
  - Network

Traces scale

- Business application can be extremly big!

- Business application traces example:
  - 20 000 traces during start-up phase (first minute)
  - 2 600 traces/min on average
  - 1h of application running = 20MB of traces (160k lines of traces)

- If problem detected in application is RARE, you will get traces only ONCE!

Your traces has to be best quality possible

Good & wrong Traces

Definitions

- Good trace:
  - Trace that gives user as most useful information as possible
- Good tracing:
  - As least traces as possible, giving overall view of what happened and what was the root cause. Allow to analyse problem relatively fast without repro

- Wrong trace:
  - Trace that does NOT contain useful information or makes analysis impossible
- Wrong tracing:
  - Lots of traces which are not telling much, sometimes allows to restore callstack, but rarely to solve the problem. Require to repro in order to solve the issue

Wrong traces

- Why wrong traces are wrong?
  - It’s difficult to tell what happened in regular case
  - It’s impossible to tell what happened in rare case
- Why too much traces is wrong?
  - Too much information makes analysis difficult – it’s hard to focus on single aspect
  - Possible performance issues
- Small number of traces?
  - Unknown callstack – bad news
  - Task doing a lot, only to check if it’s active - awesome

Examples

Not enough traces

```
int calculate(int a, int b)
{
    if(0 != b)
    {
        //Very complex calculations
    }
    else
    {
        //Error
    }

    return a;
}
```

Useless trace, double tracing, no parameters

```
int calculate(int a, int b)
{
    printf("calculate\n");

    if(0 != b)
    {
        printf("calculate ok, possible to do something\n");
    }
    else
    {
        printf("calculate error, not possible\n");
    }

    return a;
}
```

Useless trace, double tracing, no parameters

```c
int calculate(int a, int b)
{
    printf("calculate\n");

    if(0 != b)
    {
        printf("calculate ok, possible to do something\n");
    }
    else
    {
        printf("calculate error, not possible\n");
    }

    return a;
}
```

```c
int calculate(int a, int b)
{
    if(0 != b)
    {
        printf("calculate ok, a=%d b=%d\n",a,b);
    }
    else
    {
        printf("calculate error, because of %d\n",b);
    }

    return a;
}
```

Traces in loop

```
int complexTask(int param)
{
    int sum = 0;

    //Very complex task, needs lots of time to calculate.
    for(int i=0; i<param; ++i)
    {
        //Calculations
        sum += param;

        printf("complexTask sum: %d for %d\n",sum,i);
    }

    return sum;
}
```

```
complexTask sum: 21 for 0
complexTask sum: 42 for 1
complexTask sum: 63 for 2
complexTask sum: 84 for 3
complexTask sum: 105 for 4
complexTask sum: 126 for 5
complexTask sum: 147 for 6
complexTask sum: 168 for 7
complexTask sum: 189 for 8
complexTask sum: 210 for 9
complexTask sum: 231 for 10
complexTask sum: 252 for 11
complexTask sum: 273 for 12
complexTask sum: 294 for 13
complexTask sum: 315 for 14
complexTask sum: 336 for 15
complexTask sum: 357 for 16
complexTask sum: 378 for 17
complexTask sum: 399 for 18
complexTask sum: 420 for 19
complexTask sum: 441 for 20
```

Traces in loop

```
int complexTask(int param)
{
    int sum = 0;

    //Very complex task, needs lots of time to calculate.
    for(int i=0; i<param; ++i)
    {
        //Calculations
        sum += param;

        if(0 == (i%10))
        {
            printf("complexTask sum: %d for %d\n",sum,i);
        }
    }

    printf("complexTask result: %d for %d\n",sum,param);

    return sum;
}
```

```
complexTask sum: 21 for 0
complexTask sum: 231 for 10
complexTask sum: 441 for 20
complexTask result: 441 for 21
```

Reduntant tracing

```
int main()
{
    bool fRet = calculateResult(3);

    if(false != fRet)
    {
        printf("main OK\n");
    }
    else
    {
        printf("main ERROR"\n);
    }

    return 0;
}
```

```
bool calculateResult(int param)
{
    printf("calculateResult\n");
    bool fRetVal = false;

    //some operations with param
    int value = param * 2;

    //call checker
    fRetVal = checkParameter(value);

    if(false != fRetVal)
    {
        printf("calculateResult OK\n");
    }
    else
    {
        printf("calculateResult ERROR\n");
    }

    return fRetVal;
}
```

```
calculateResult
checkParam ERROR
calculateResult ERROR
main ERROR
```

```
bool checkParameter(int param)
{
    bool fRetVal = true;

    if(param < 10)
    {
        printf("checkParam ERROR\n");
        fRetVal = false;
    }

    return fRetVal;
}
```

Reduntant tracing

```
int main()
{
    bool fRet = calculateResult(3);

    return 0;
}
```

```
bool calculateResult(int param)
{
    bool fRetVal = false;

    //some operations with param
    int value = param * 2;

    //call checker
    fRetVal = checkParameter(value);

    if(false != fRetVal)
    {
        printf("calculateResult OK for %d\n",param);
    }
    else
    {
        printf("calculateResult ERROR for %d\n",param);
    }

    return fRetVal;
}
```

```
checkParam ERROR for 6
calculateResult ERROR for 3
```

```
bool checkParameter(int param)
{
    bool fRetVal = true;

    if(param < 10)
    {
        printf("checkParam ERROR\n");
        fRetVal = false;
    }

    return fRetVal;
}
```

Summary

What should be traced?

- Input values
- Summaries
- Success
- Warnings
- Errors (decide on which function nest level)

- Member values if used in function
- Thread prio, threads Ids at startup

What should NOT be traced

- Empty function entries, unless necessary - try to make it useful
- Loops and every calculation
- „Timed" events if too much

- It all depends on the situation – think, try, rework. Be smart!

- Collect traces while testing your code!
  - Are you able to tell what happened?
  - Which traces are not needed?

Traces example - OLD

```
int main(int argc, char* argv[])
{
    printf("Example calculations\n");

    int counter = 10;
    printf("Start calculation for %d\n",counter);

    int sum = 0;
    for(int i=0; i<counter; ++i)
    {
        sum = sum + 2;
        printf("Iteration %d/%d, temp sum = %d\n",i,counter,sum);
    }

    printf("Result = %d\n",sum);

    return 0;
}
```

```
Example calculations
Start calculation for 10
Iteration 0/10, temp sum = 2
Iteration 1/10, temp sum = 4
Iteration 2/10, temp sum = 6
Iteration 3/10, temp sum = 8
Iteration 4/10, temp sum = 10
Iteration 5/10, temp sum = 12
Iteration 6/10, temp sum = 14
Iteration 7/10, temp sum = 16
Iteration 8/10, temp sum = 18
Iteration 9/10, temp sum = 20
Calc for 10 = 20
```

Traces example - NEW

```
int main(int argc, char* argv[])
{
    //printf("Example calculations\n");

    int counter = 10;
    //printf("Start calculation for %d\n",counter);

    int sum = 0;
    for(int i=0; i<counter; ++i)
    {
        sum = sum + 2;
        //printf("Iteration %d/%d, temp sum = %d\n",i,counter,sum);
    }

    printf("Calc for %d = %d\n",counter, sum);

    return 0;
}
```

```
Calc for 10 = 20
```

Additional hints

Hints for good traces

- Add some more traces for new modules, remove from old modules
- Avoid the use of **'decorators'** (e.g. ****, !!!)

- Read documentation
  - Some functions may return error which needs to be printed with **GetLastError**() [WinAPI]
  - We may not want to support every error – single trace can be used for different errors

WaitForSingleObject [WinAPI]

Return values example

Return code/value	Description
WAIT_ABANDONED 0x00000080L	The specified object is a mutex object that was not released by the thread that owned the mutex object before the owning thread terminated. Ownership of the mutex object is granted to the calling thread and the mutex state is set to nonsignaled. If the mutex was protecting persistent state information, you should check it for consistency.
WAIT_OBJECT_0 0x00000000L	The state of the specified object is signaled.
WAIT_TIMEOUT 0x00000102L	The time-out interval elapsed, and the object's state is nonsignaled.
WAIT_FAILED (DWORD)0xFFFFFFFF	The function has failed. To get extended error information, call **GetLastError**.

I need very looooooooooong trace

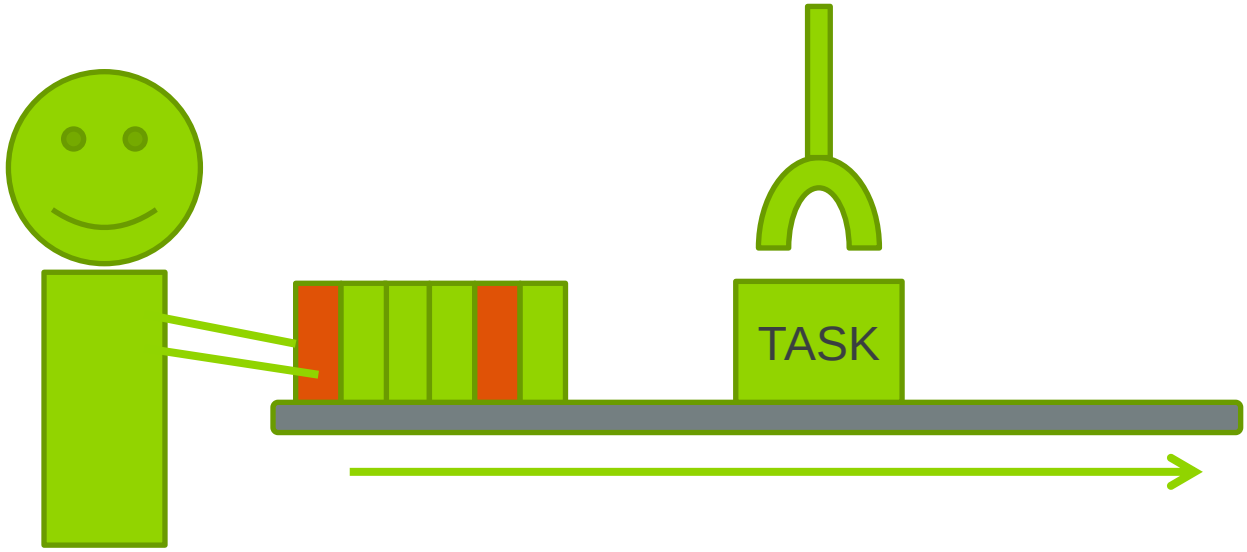
- Compress it into short one!

- Example:
  - CMyClassObject::myVeryLongFunctionName parameter1=2, parameter2=8, parameter3=15
  - CMCO::myVeryLongFncName p1:2 p2:8 p3:15

- Keep identifier unique
- Keep messages **as short as possible** without making them unreadable

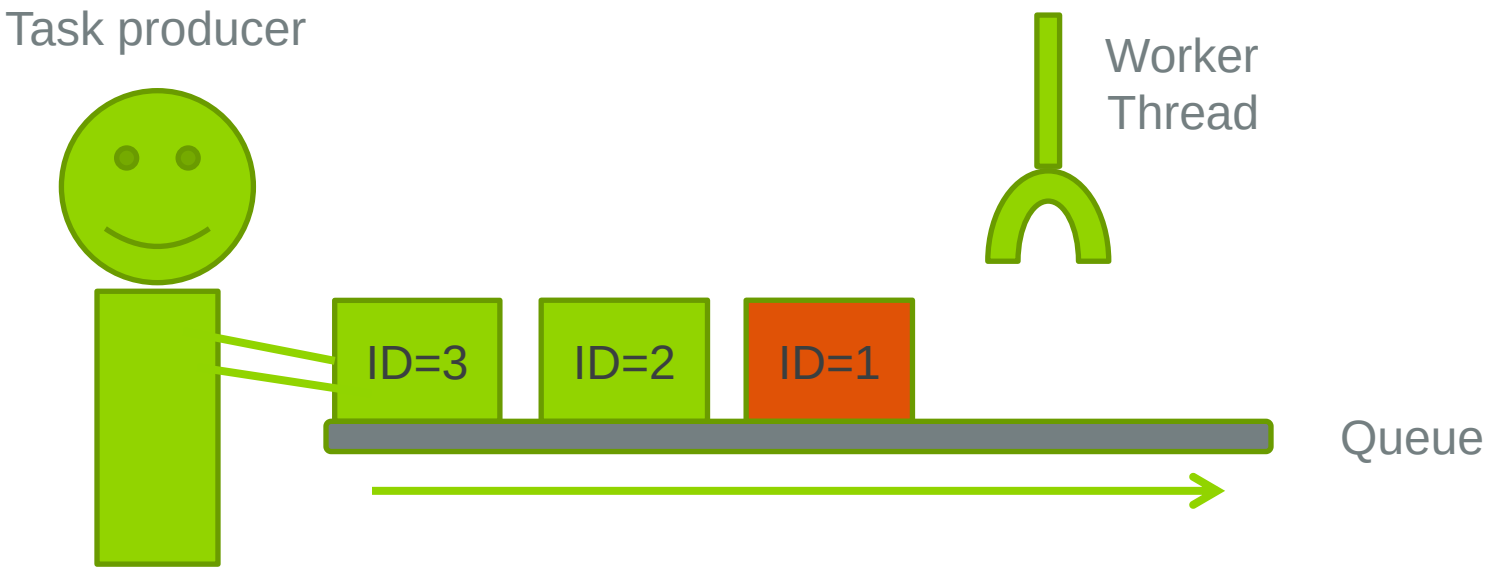
Advanced example

Advanced example



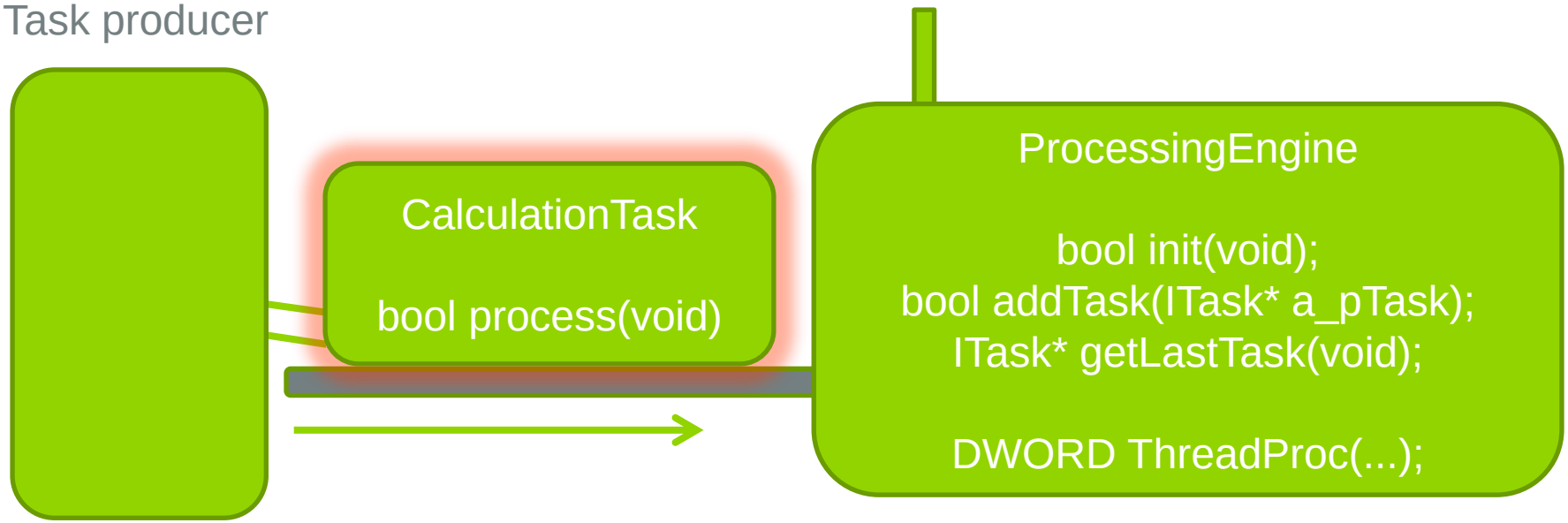
- Producer add several tasks to queue
- Some of the tasks are „problematic” and cannot be calculated
- Machine process tasks, one after another & tries to calculate result

Advanced example



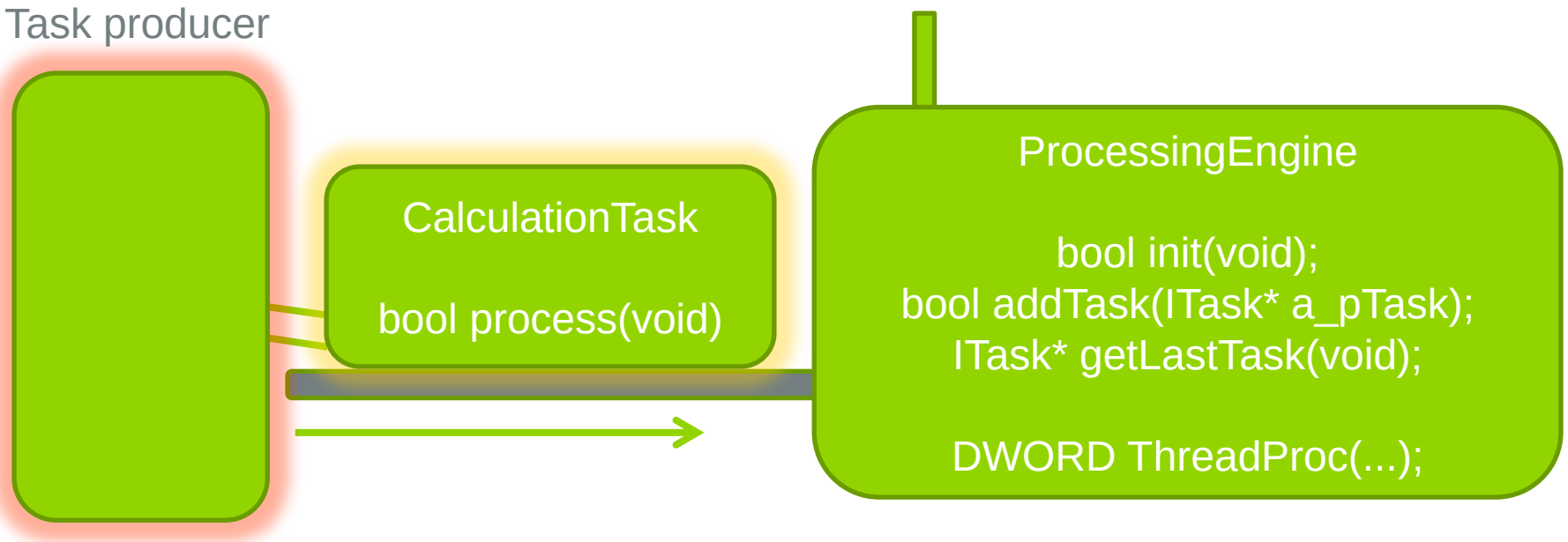
- Task producer adds task objects to Queue
- Queue handles tasks
- Worker Thread is waiting for task to process

Advanced example



- CalculationTask:
  - calculates result which takes some time (ID * 3 * 1000msec)
  - return true on success or false on fail

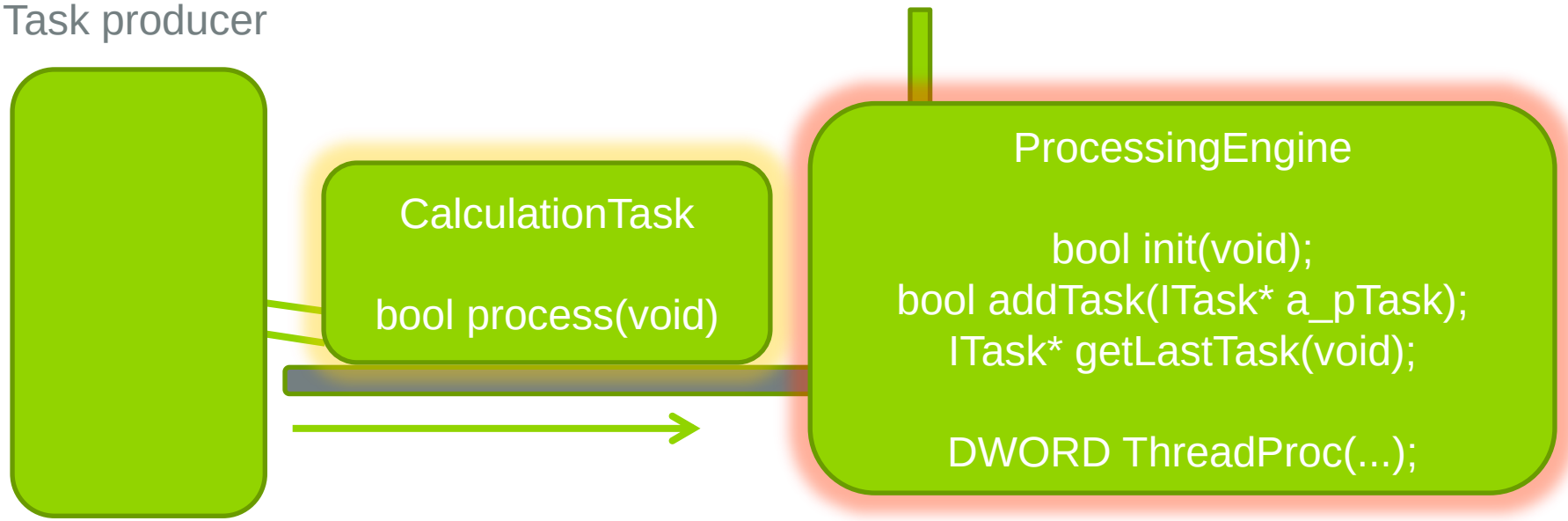
Advanced example



- Task producer adds 3 tasks to ProcessingEngine queue:

ID=3 success	ID=2 success	ID=1 fail

Advanced example



- ProcessingEngine:
  - when not busy, takes first task from queue (removing it)
  - process taken task
  - wait for another task

Advanced example – trace output

- SAME logic – 1 BUG

```
void wrongExample()
{
    //START engine
    CWrongProce
    oEngine.ini
    Sleep(500);

    //Add tasks
    CWrongCalcu
    CWrongCalcu
    CWrongCalcu
    oEngine.add
    oEngine.add
    oEngine.add

    return;
}
```

```
void goodExample()
{
    //START engine
    CGoodProcessingEngine oEngine;
    oEngine.init();
    Sleep(500); //Let the thread start

    //Add tasks to engine to process them
    CGoodCalculationTask* task1 = new CGoodCalculationTask(1, false);
    CGoodCalculationTask* task2 = new CGoodCalculationTask(2, true);
    CGoodCalculationTask* task3 = new CGoodCalculationTask(3, true);
    oEngine.addTask(task1);
    oEngine.addTask(task2);
    oEngine.addTask(task3);

    return;
}
```

Wrong ProcessingEngine

```
CWrongProcessingEngine::init
CWrongProcessingEngine::ThreadProc
CWrongProcessingEngine::addTask
CWrongProcessingEngine::addTask
CWrongProcessingEngine::addTask
CWrongCalculationTask::process
CWrongCalculationTask::process calculation in progress
CWrongCalculationTask::process calculation in progress
CWrongCalculationTask::process calculation in progress
CWrongProcessingEngine::ThreadProc process FAIL
CWrongCalculationTask::process
CWrongCalculationTask::process calculation in progress
CWrongCalculationTask::process calculation in progress
CWrongCalculationTask::process calculation in progress
CWrongCalculationTask::process calculation in progress
CWrongCalculationTask::process calculation in progress
CWrongCalculationTask::process calculation in progress
CWrongCalculationTask::getCalculatedResult is 12
CWrongProcessingEngine::ThreadProc process OK calculation result is 12
```

Good ProcessingEngine

```
[INF] CGE::init on 0x18ff24
[INF] CGE::ThreadProc Prio:0 on 0x18ff24
[INF] CGE::addTask ID:1 size:1 Added
[INF] CGE::addTask ID:2 size:1 Added
[INF] CGE::addTask ID:3 size:2 Added
[INF] CGT::process ID:1 START
[ERR] CGT::process ID:1 FAIL 3042ms
[INF] CGT::process ID:2 START
[INF] CGT::process ID:2 DONE 6084ms
[INF] CGE::ThreadProc process OK for ID:2 Res:12
```

Comparison

```
CWrongProcessingEngine::init
CWrongProcessingEngine::ThreadProc
CWrongProcessingEngine::addTask
CWrongProcessingEngine::addTask
CWrongProcessingEngine::addTask
CWrongCalculationTask::process
CWrongCalculationTask::process calculation in progress
CWrongCalculationTask::process calculation in progress
CWrongCalculationTask::process calculation in progress
CWrongProcessingEngine::ThreadProc process FAIL
CWrongCalculationTask::process
CWrongCalculationTask::process calculation in progress
CWrongCalculationTask::process calculation in progress
CWrongCalculationTask::process calculation in progress
CWrongCalculationTask::process calculation in progress
CWrongCalculationTask::process calculation in progress
CWrongCalculationTask::pro
CWrongCalculationTask::get
CWrongProcessingEngine::Th
```

```
[INF] CGE::init on 0x18ff24
[INF] CGE::ThreadProc Prio:0 on 0x18ff24
[INF] CGE::addTask ID:1 size:1 Added
[INF] CGE::addTask ID:2 size:1 Added
[INF] CGE::addTask ID:3 size:2 Added
[INF] CGT::process ID:1 START
[ERR] CGT::process ID:1 FAIL 3042ms
[INF] CGT::process ID:2 START
[INF] CGT::process ID:2 DONE 6084ms
[INF] CGE::ThreadProc process OK for ID:2 Res:12
```

Conclusion

- Make Traces:
  - Concise
  - Readable
  - Matter

„think, try, rework. Be smart!”

Questions?

Marcin Mrowiński

Senior Software Engineer

brightONE

marcin.mrowinski@brightone.pl