

映像制作パイプラインにおけるPDGの役割

Pavel Smirnov

株式会社グリオグループ・LiNDAチーム

PDGとは

凄そうだけど、なんか今一、、？

使い道

- プロシージャルコンテンツ作り
- レンダーファームの扱いをより効率化
- ウェッジング
- パイプライン
- Houdiniコミュニティがこれから発見する使い方

使い道

- プロシージャルコンテンツ作り
- レンダーファームの扱いをより効率化
- ウェッジング
- **パイプライン** ←
- Houdiniコミュニティがこれから発見する使い方

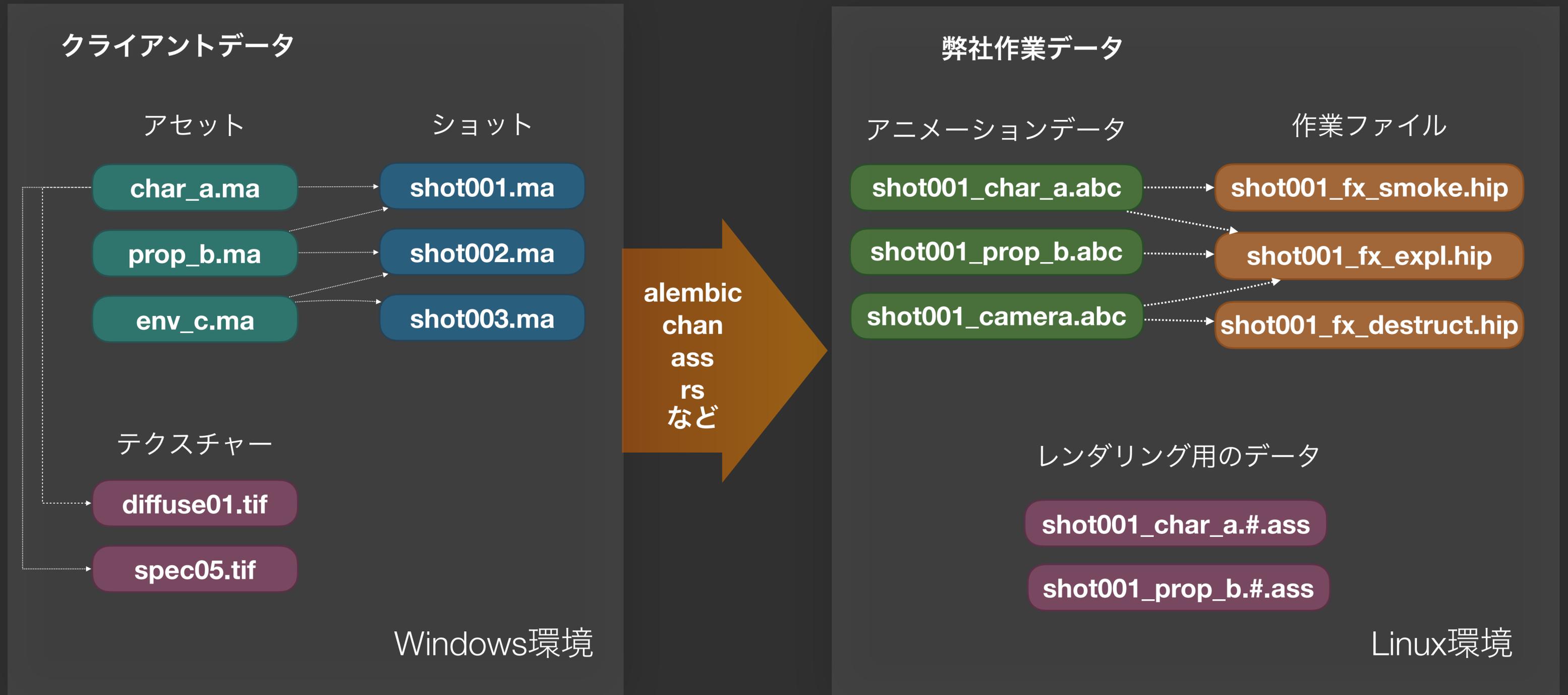
注意

以前Pythonが書けないと出来なかったことが簡単になってきた

- 一瞬でファームに500本のシミュレーションジョブを投げる
- ファイルを分散コピーしすぎて、サーバーを落とす
- 短期でデータをウン十テラ作っちゃう
- Mayaファイルを探して全て消す

本気で実行する前、
必ずテストデータに使ってみよう

よくあるパイプラインタスク：データ変換と読み込み



問題

- WinとLinuxのパス
- リファレンス再読み込み
- テクスチャーパス
- 個別に書き出すものが多い
- データが足りなかった場合の対応
- ショットの数が多ければ、手作業では無理

ROPベースパイプライン

Outputs



Maya maya1

Scene `/usr/autodesk/maya2018/Examples/Modeling/Sculpting_Base_Meshes/Bipeds/HumanBody.ma`

Open in

- Export Alembic
- Export Render Proxies**
- Export Camera
- Export Materials
- Export Lights
- Export Playblast
- Execute Custom Script**
- Refresh On Render

Refresh

Maya Command `/usr/autodesk/maya2018/bin/maya`

Alembic **Render Proxies** Camera Materials Lights Playblast Custom Script **Maya** Flipbook

Proxies Globals

Arnold

- Export ASS**
- Output Folder `$HIP/ass/$OS`
- Global Options

Redshift

- Export RS
- Output Folder `$HIP/rs/$OS`

Objects Export Setup

Arnold File Expression ``chs("assdir")`/$OBJNAME/$OBJNAME.ass`

Redshift File Expression ``chs("rsdir")`/$OBJNAME/$OBJNAME.rs`

Copy From Alembic

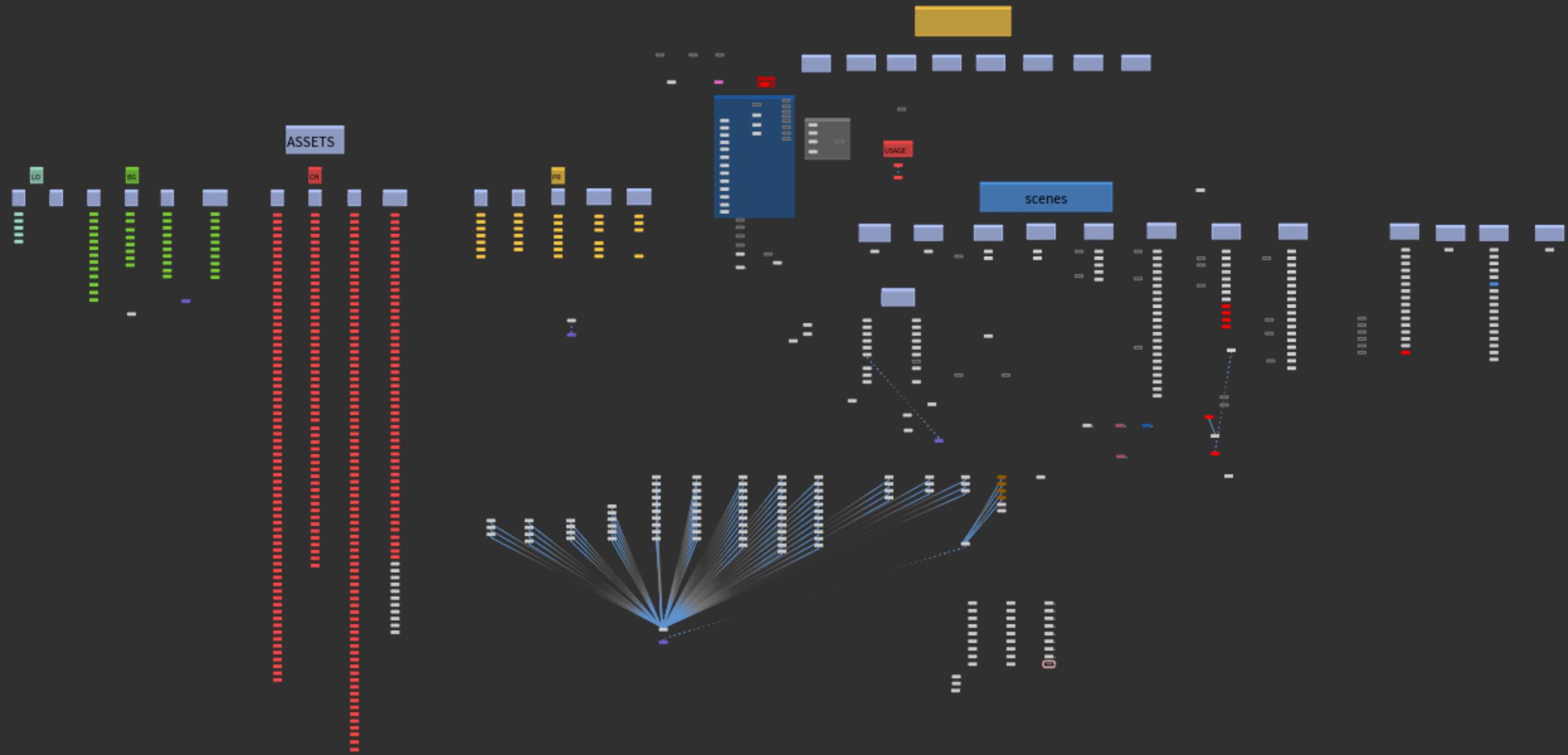
Objects `1` + - Clear

Object 0

- Enable
- Object Path `humanBody`
- ASS File ``chs("assdir")`/humanBody/humanBody.ass`
- RS File `chs("rsdir")`/humanBody/humanBody.rs`
- Arnold Options

Input Operators (Drag/Drop to reorder)

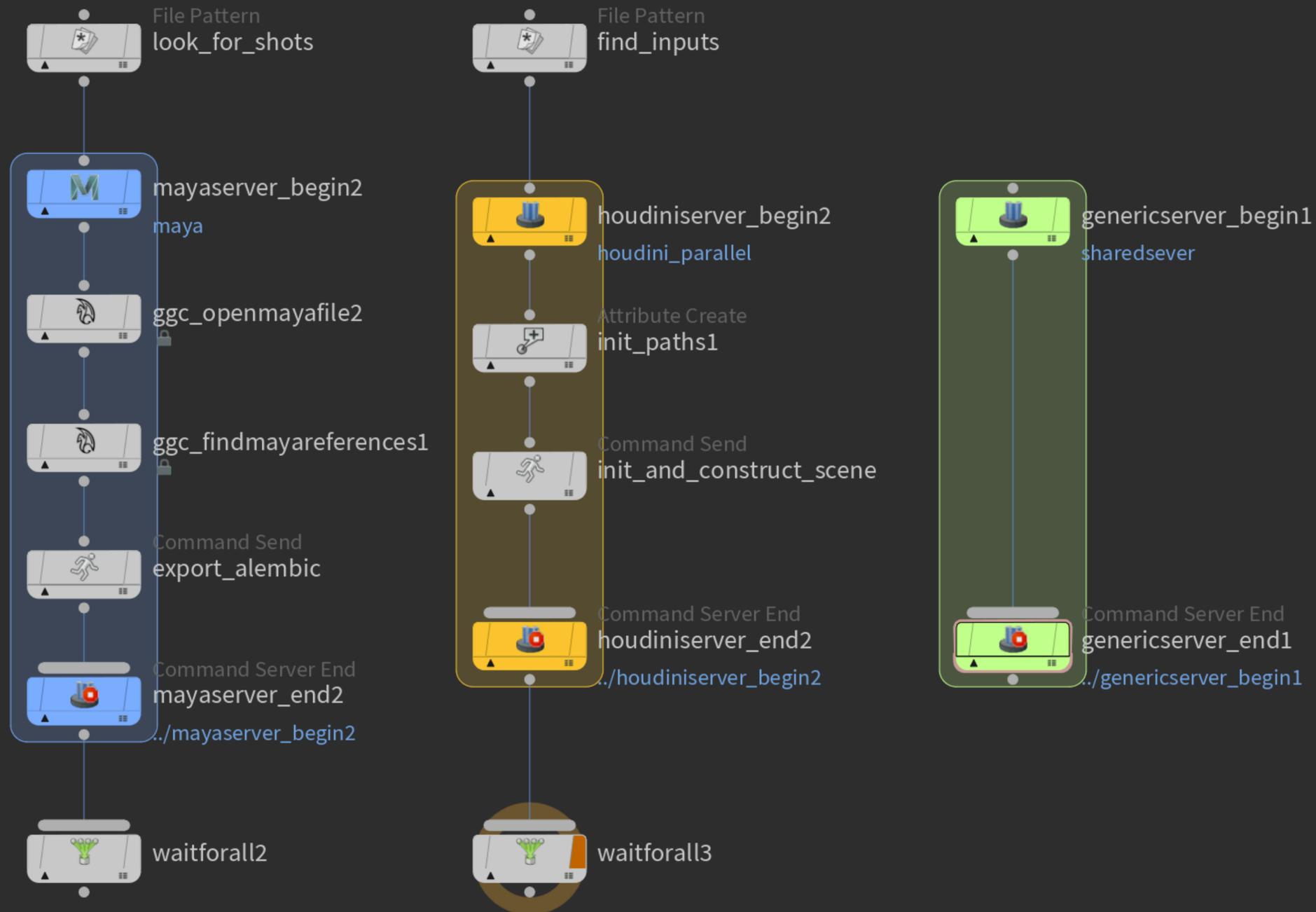
あるプロジェクトの管理シーン



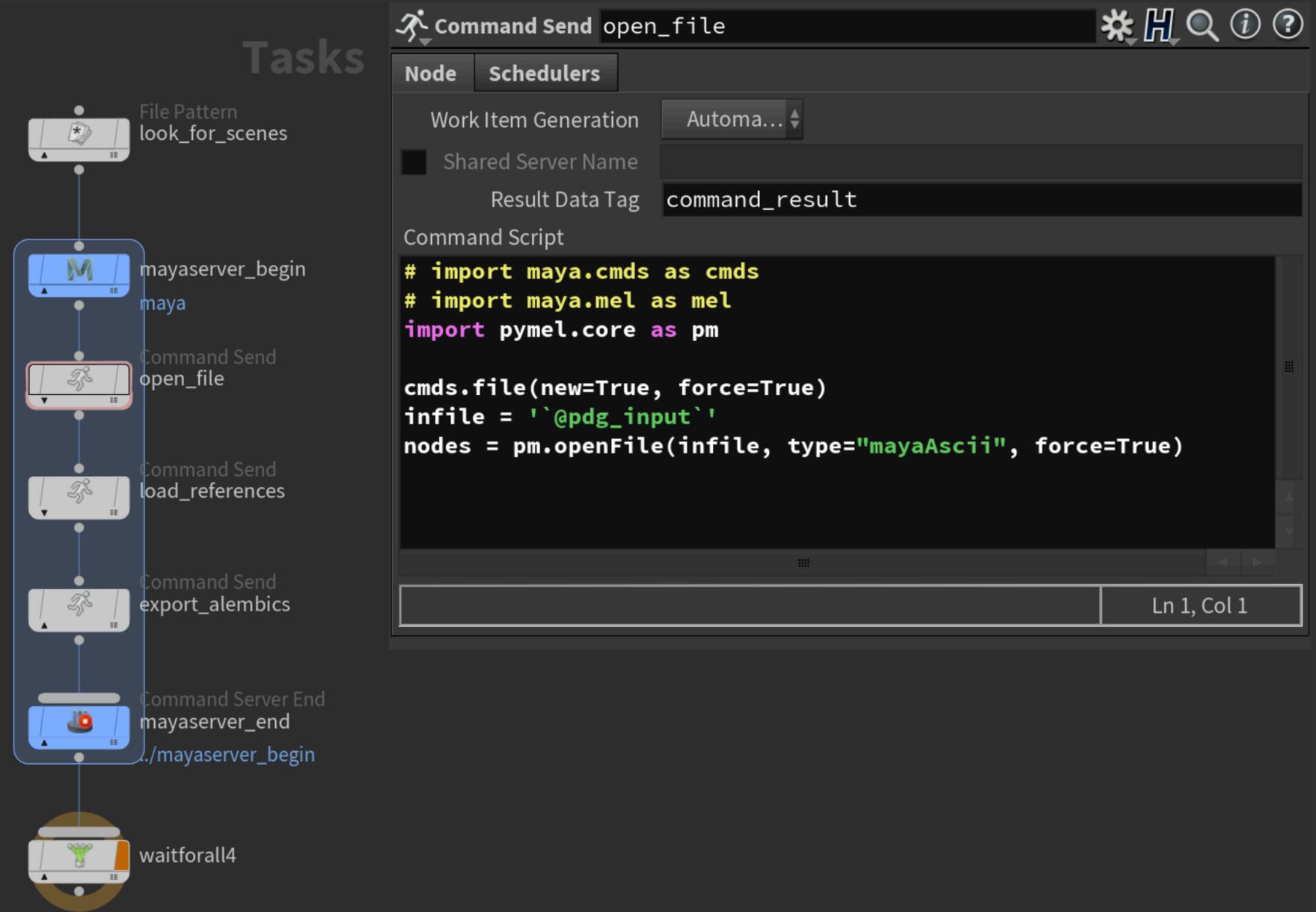
ROPベースパイプラインのデメリット

- Pythonヘビー
- カスタムROPのデバッグやアップデートに会社のリソースをかけなければならない
- 一つのシーンの扱いはプロシージャルではない
- フレキシビリティが足りない
- 大きめのプロジェクトの管理シーンは恐ろしくなる傾向がある
- 最低Houdini Coreのライセンスが必要

Command Chains



Maya Command Chain



Command Send open_file

Node Schedulers

Work Item Generation Automa...

Shared Server Name

Result Data Tag command_result

Command Script

```
# import maya.cmds as cmds
# import maya.mel as mel
import pymel.core as pm

cmds.file(new=True, force=True)
infile = '@pdg_input'
nodes = pm.openFile(infile, type="mayaAscii", force=True)
```

Ln 1, Col 1

Maya Command Chain

ディスクでMayaシーンを探して
ルートワークアイテムを作る

これがワークアイテム

The screenshot displays the Maya Command Chain interface. On the left, a vertical chain of tasks is shown, with the top task, 'File Pattern look_for_scenes', highlighted by a red box and a green checkmark. A dotted orange line points from the text 'これがワークアイテム' to this task. Below it, a blue box encloses a sequence of tasks: 'mayaserver_begin maya', 'Command Send open_file', 'Command Send load_references', 'Command Send export_alembics', and 'Command Server End mayaserver_end ./mayaserver_begin'. At the bottom of the chain is a 'waitforall4' task. On the right, the configuration panel for the 'File Pattern look_for_scenes' task is visible, showing settings for Work Item Generation (Automated), File Types (Files Only), Glob Pattern (\$JOB/assets/chars/*/scenes/*.ma), Recursive (unchecked), Result Data Tag (file/ma), Work Item Index (0), Index From File Sequence (unchecked), Split Results into Separate Items (checked), and Error on No Matches (unchecked).

Maya Command Chain

裏でMayaを立ち上げて

The image displays the Maya Command Chain interface. On the left, a vertical task flow is shown with the following steps:

- 4 ● ✓ File Pattern look_for_scenes
- 2 ● 1 ● 9 ● **mayaserver_begin** (highlighted with a red box)
- 1 ● 3 ● Command Send open_file
- 4 ● Command Send load_references
- 4 ● Command Send export_alembics
- 4 ● Command Server End mayaserver_end
- 1 ● waitforall4

On the right, the 'Command Send open_file' task is expanded, showing the following configuration:

- Node: open_file
- Schedulers: Automa...
- Work Item Generation: Automa...
- Shared Server Name: (empty)
- Result Data Tag: command_result
- Command Script:

```
# import maya.cmds as cmds
# import maya.mel as mel
import pymel.core as pm

cmds.file(new=True, force=True)
infile = '@pdg_input'
nodes = pm.openFile(infile, type="mayaAscii", force=True)
```
- Ln 3, Col 1

Maya Command Chain

各アイテムに対して
MayaにMELかPythonの
スクリプトを送って実行する

The image shows the Maya Command Chain interface. On the left, a vertical list of tasks is displayed under the heading "Tasks". The tasks are:

- File Pattern look_for_scenes (4 items, green checkmark)
- mayaserver_begin maya (4 items, 1 green dot, 7 grey dots)
- Command Send open_file (1 item, 3 grey dots) - This task is highlighted with a red box.
- Command Send load_references (4 items, 3 grey dots)
- Command Send export_alembics (4 items, 3 grey dots)
- Command Server End mayaserver_end (4 items, 3 grey dots) with sub-task ./mayaserver_begin
- waitforall4 (1 item, 3 grey dots)

On the right, a detailed view of the "Command Send open_file" task is shown. The "Node" is "Command Send" and the "Schedulers" are "open_file". The "Work Item Generation" is set to "Automa...". The "Result Data Tag" is "command_result". The "Command Script" is:

```
# import maya.cmds as cmds
# import maya.mel as mel
import pymel.core as pm

cmds.file(new=True, force=True)
infile = '@pdg_input'
nodes = pm.openFile(infile, type="mayaAscii", force=True)
```

The status bar at the bottom right indicates "Ln 3, Col 1".

Maya Command Chain

枠の中のノードはMayaの中で行なっている

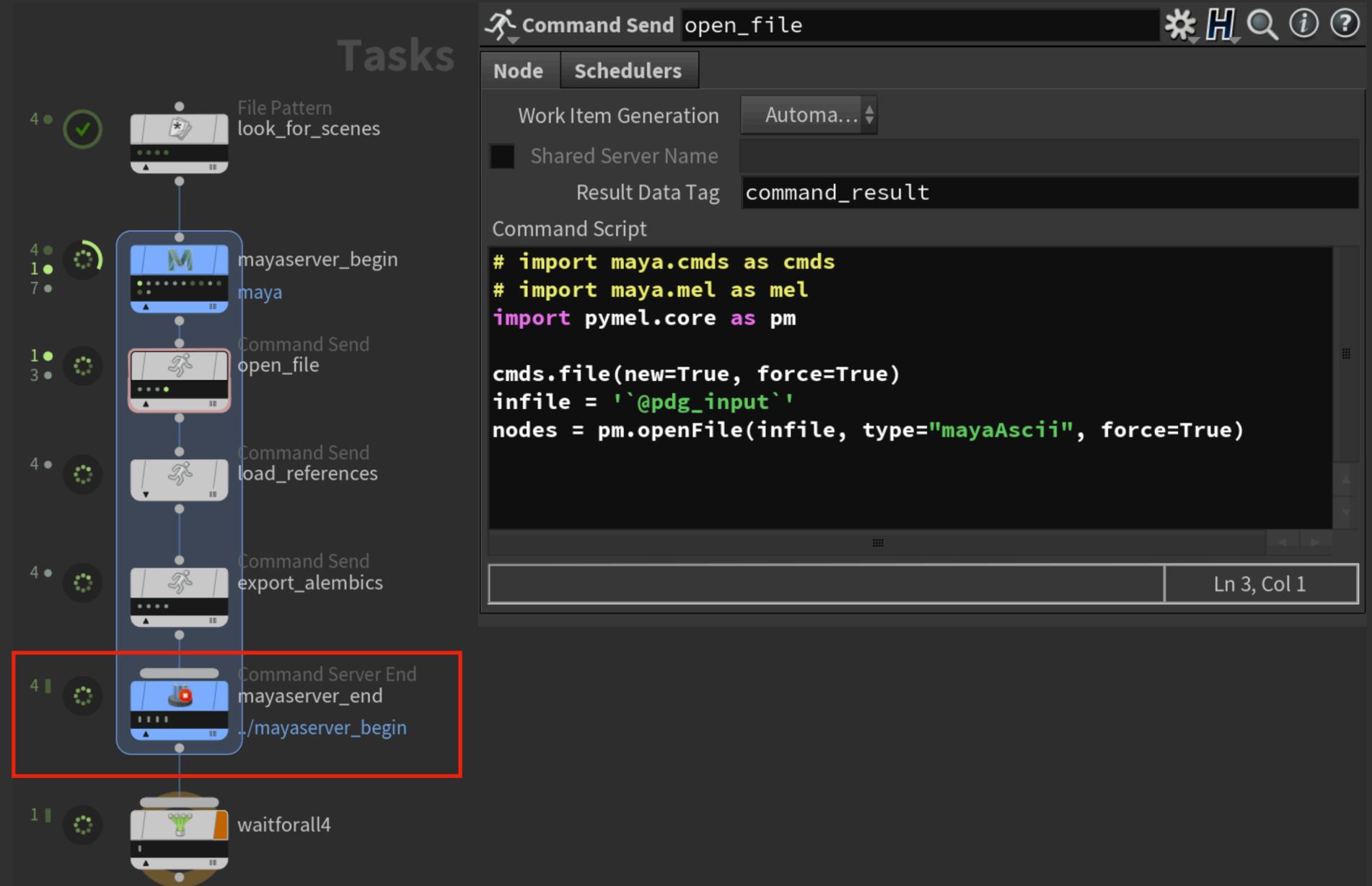
The image displays the Maya Command Chain interface. On the left, a vertical task flow is shown with nodes: File Pattern look_for_scenes, mayaserver_begin maya, Command Send open_file, Command Send load_references, Command Send export_alembics, Command Server End mayaserver_end ./mayaserver_begin, and waitforall4. A red box highlights the 'Command Send open_file' node. On the right, a detailed view of the 'Command Send open_file' node is shown. The 'Node' tab is active, displaying the following command script:

```
# import maya.cmds as cmds
# import maya.mel as mel
import pymel.core as pm

cmds.file(new=True, force=True)
infile = '@pdg_input'
nodes = pm.openFile(infile, type="mayaAscii", force=True)
```

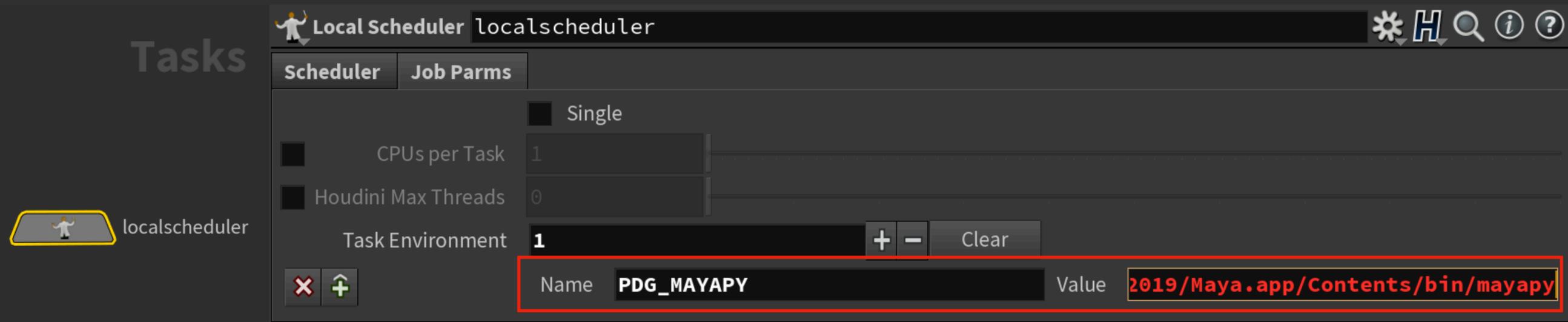
The 'Schedulers' tab is also visible, showing 'Work Item Generation' set to 'Automa...', 'Shared Server Name' as an empty field, and 'Result Data Tag' set to 'command_result'. The status bar at the bottom right indicates 'Ln 3, Col 1'.

Maya Command Chain



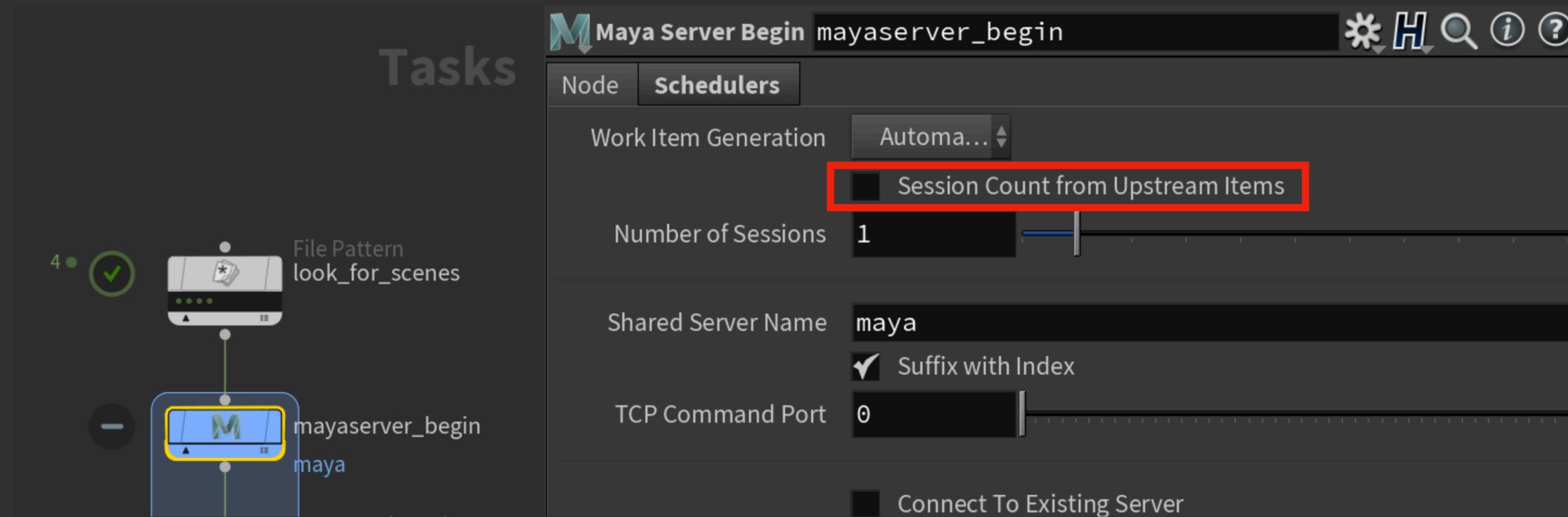
結果をまとめて
Mayaを落とす

大事な設定：PDG_MAYAPY 環境変数



- Linux: `/usr/autodesk/maya2019/bin/mayapy`
- Mac: `/Applications/Autodesk/maya2019/Maya.app/Contents/bin/mayapy`
- Windows: `C:/Program Files/Autodesk/Maya2019/bin/mayapy.exe`

大事な設定：Session Count From Upstream Items



- **OFF:** 各アイテムに別のMayaのプロセスを立ち上げる

例：Maya起動→Open→作業→Save→Quit

Maya起動→New→作業→Save→Quit

Maya起動→Open→作業→Save→Quit

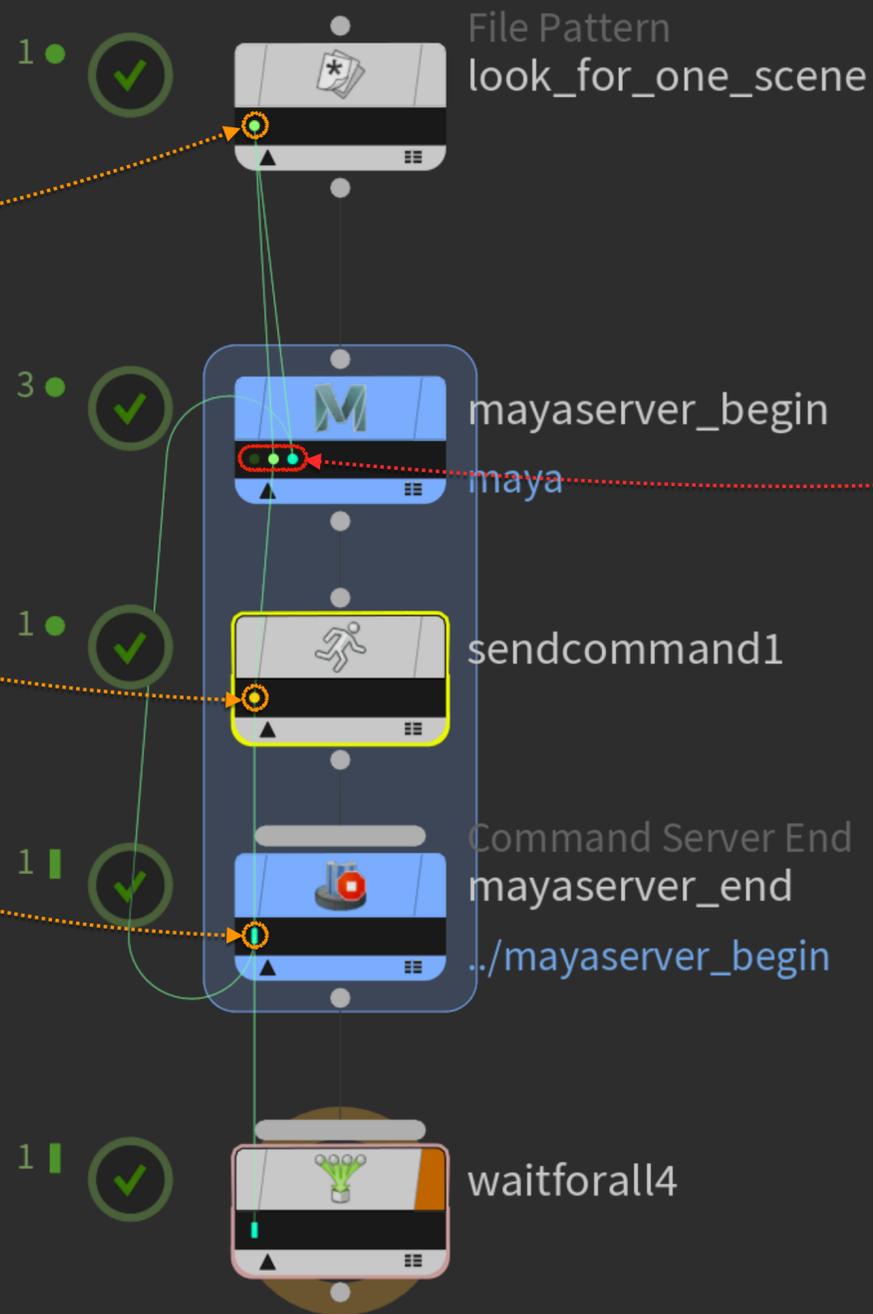
} 同時にできる

- **ON:** 全てのアイテムを順番で一つのプロセスでやる

例：Maya起動→Open→作業→Save→Open→作業→Save→New→作業 など

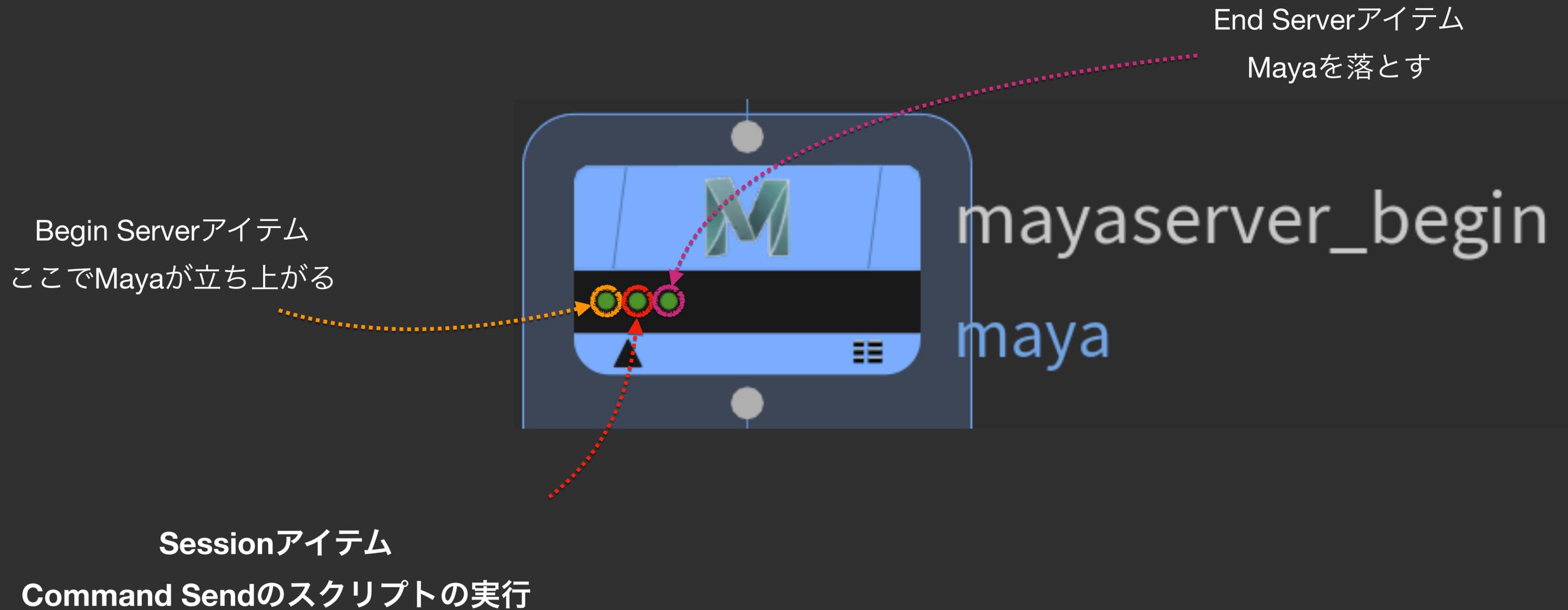
マニアックのコーナー：Server Items

一個づつ
(通常通り)



なんでここだけ三つ??

マニアックのコーナー：Server Items

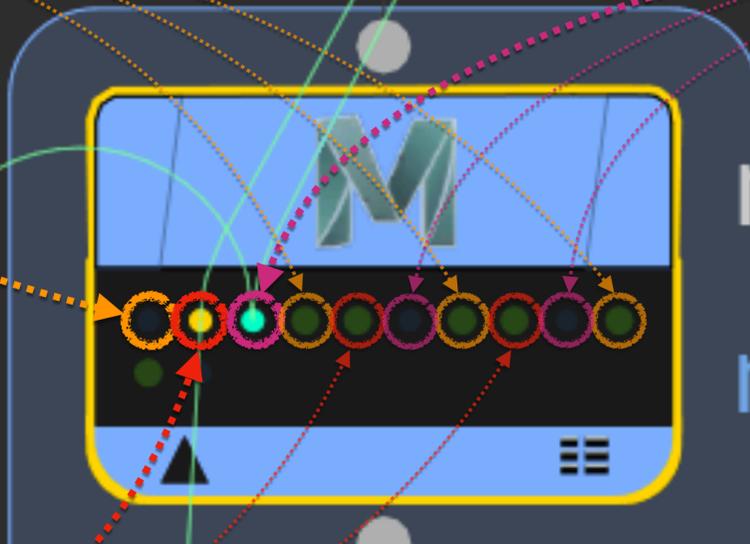


マニアックのコーナー：Server Items

■ Session Count from Upstream Items

Mayaを立ち上げて

Mayaを落とす



mayaserver_be
maya

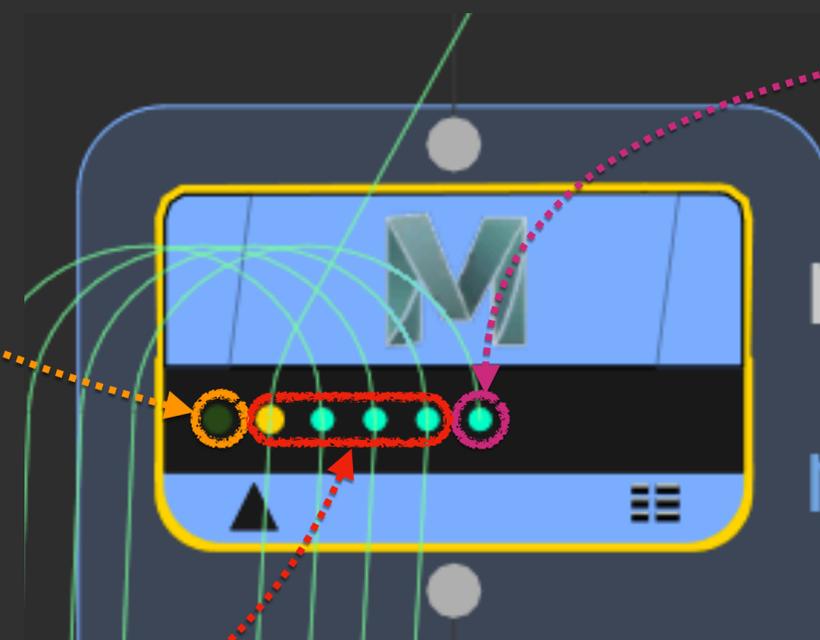
別々のMayaでスクリプト実行

マニアックのコーナー：Server Items

Session Count from Upstream Items

Mayaを立ち上げて

Mayaを落とす



mayaserver_be

maya

セッションのアイテム
共有のMayaでスクリプト実行

大事な設定：Local Scheduler Job Parameters

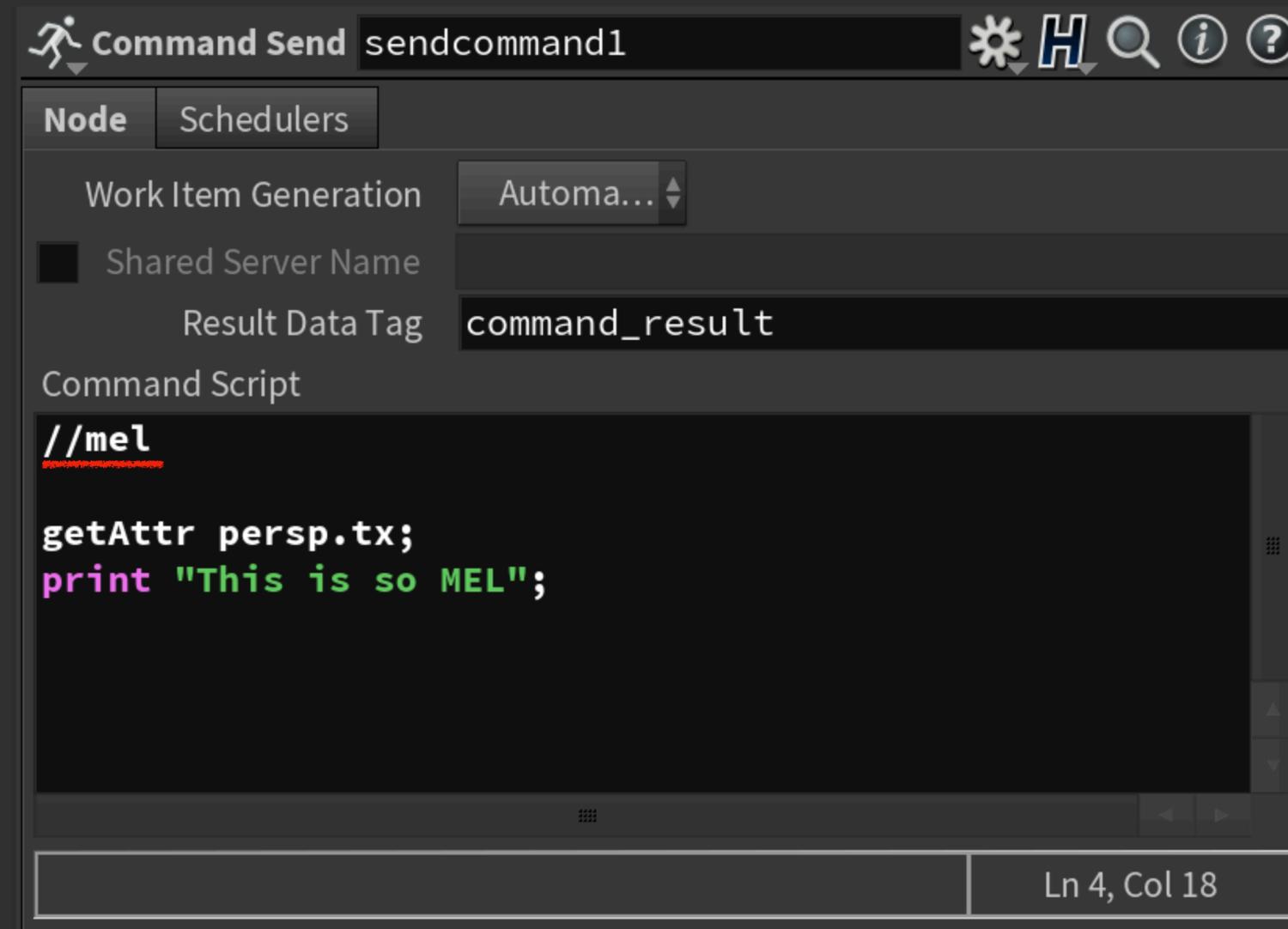
The image shows a screenshot of the Houdini interface. On the left, a task graph is visible with a node named 'mayaserver_begin' highlighted in blue. The main window displays the 'Local Scheduler' settings for the 'mayaserver_begin' node. The 'Job Parameters' section is expanded, showing the following settings:

- Override Job Parameters
- Single
- CPUs per Task: 1
- Houdini Max Threads: 0
- Task Environment: 0

The 'Override Job Parameters' and 'Single' checkboxes are highlighted with red boxes. The 'Task Environment' field has a 'Clear' button next to it.

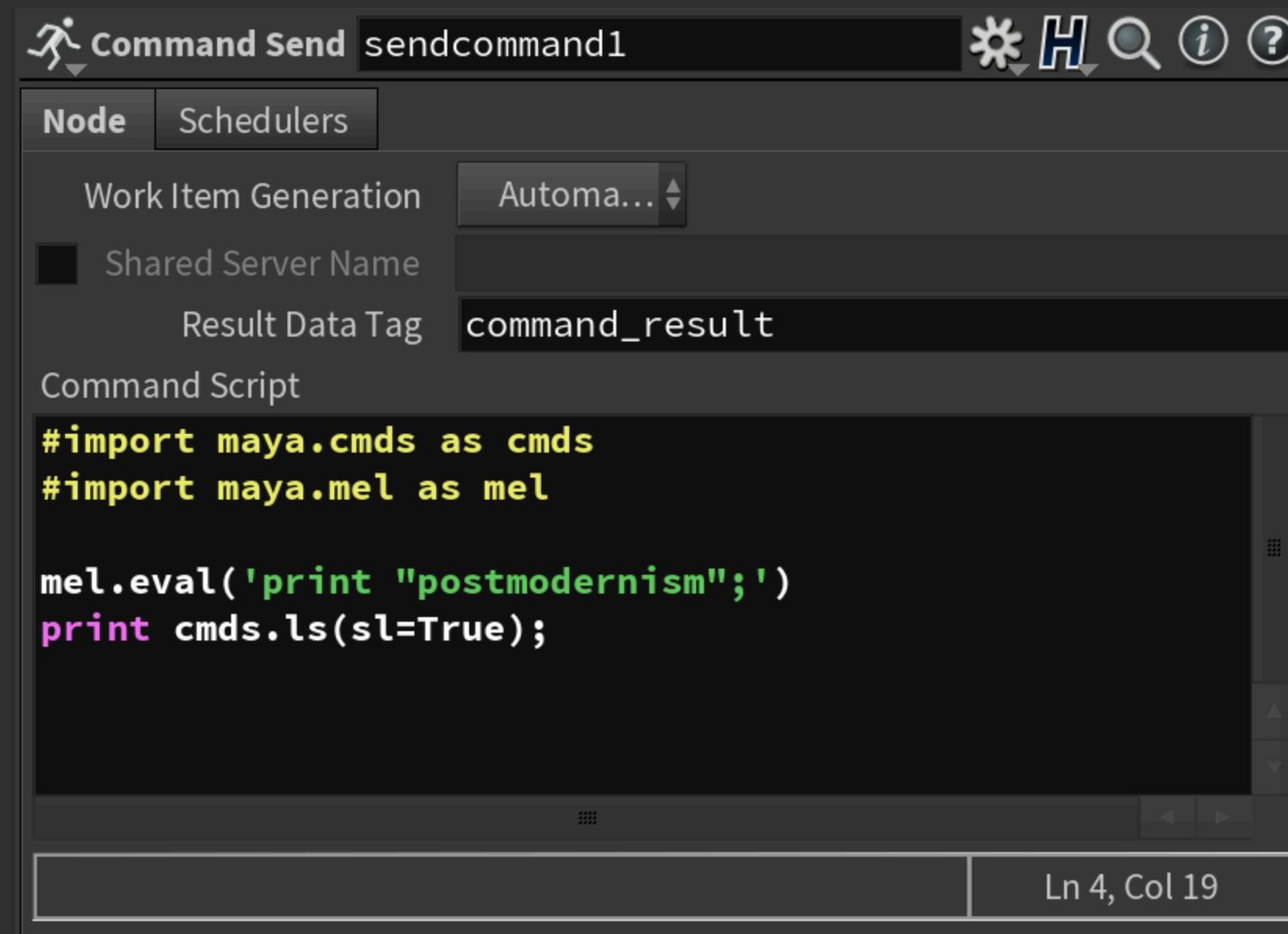
重いシーンを扱う場合、同時にMayaを一本しか立ち上がらないようにする設定

Command Send: MEL



MELスクリプトを書く際、一番最初に **//mel** を書かなければならない

Command Send: Python



- //melを入れなければ、Pythonになる
- maya.cmdsとmaya.melは自動インポートされている

Command Send: Logs

The screenshot shows the Maya Command Send interface. On the left, the 'Command Script' field contains the following Python code:

```
//mel  
print "LET'S PRINT SOME NICE USER LOGS";
```

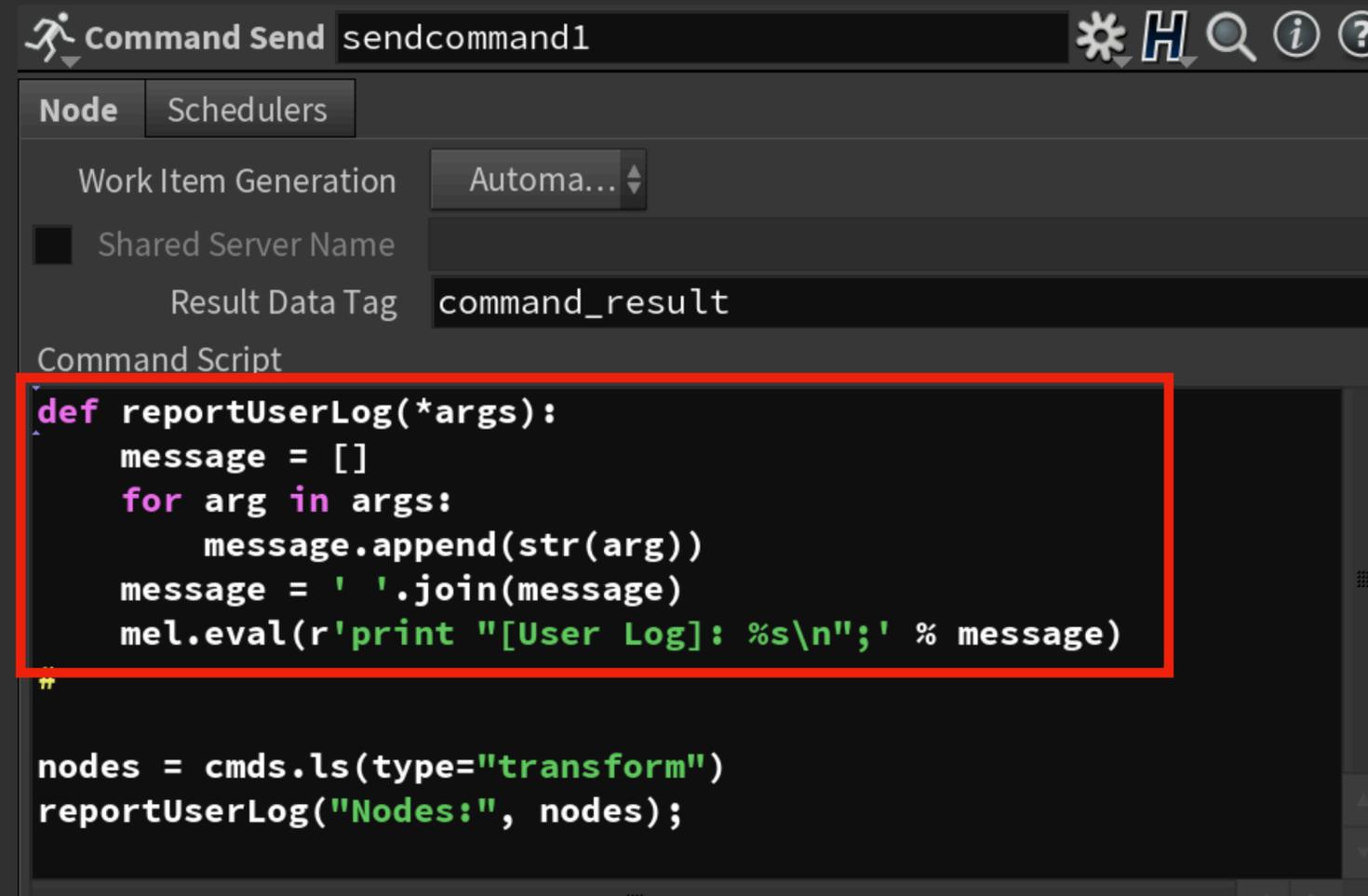
In the center, a terminal window titled 'mayaserver_begin_beginserver_4' displays the following output:

```
mayaserver_begin  
mayaserver_begin_beginserver_4  
Task  
State Finished  
Index 0  
Priority 0  
No Generate True  
Command "__PDG_PYTHON__" "__PDG_SCRIPTDIR__/sharedserver.py" --start --name  
mava0 --port 0 --timeout 30 "$PDG_MAYAPY" " PDG_SCRIPTDIR /  
Hide log  
INFO:__main__:Attempting to connect to sharedserver at ('MacBook-P  
This plugin does not support createPlatformOpenGLContext!  
PDG_RESULT: mayaserver_begin_beginserver_4;-1;'MacBook-Pro.local';  
PDG_RESULT: mayaserver_begin_beginserver_4;-1;'59131';socket/port;  
LET'S PRINT SOME NICE USER LOGS
```

On the right, a workflow graph shows several nodes. The 'mayaserver_begin' node is highlighted with a yellow box. A red arrow points from the terminal window to this node, with the text 'print出力' (print output). Another red arrow points from the 'mayaserver_begin' node to the text 'エラー以外のログは Begin Serverのアイテムに入る' (Logs other than errors go into the Begin Server item).

※私のPython知識不足か、Pythonのprintはアイテムのログに出てこない

Command Send: python log



The screenshot shows the Command Send interface with the following configuration:

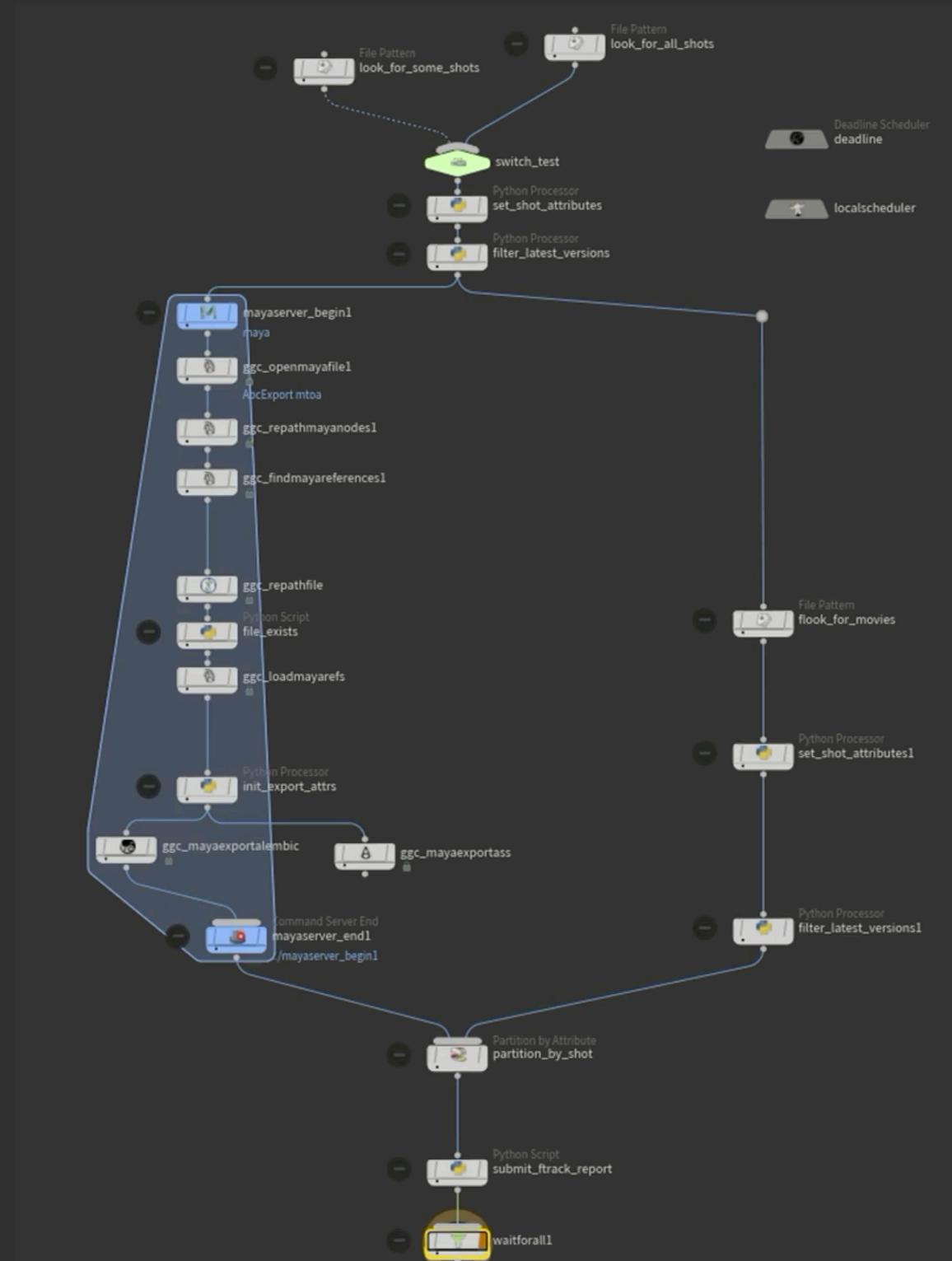
- Node: sendcommand1
- Work Item Generation: Automa...
- Shared Server Name: (empty)
- Result Data Tag: command_result
- Command Script:

```
def reportUserLog(*args):  
    message = []  
    for arg in args:  
        message.append(str(arg))  
    message = ' '.join(message)  
    mel.eval(r'print "[User Log]: %s\n";' % message)  
  
nodes = cmds.ls(type="transform")  
reportUserLog("Nodes:", nodes);
```

[Hide log](#)

```
INFO:__main__:Attempting to connect to sharedserver at ('MacBook-Pro.local', 59593)  
This plugin does not support createContext!  
PDG_RESULT: mayaserver_begin_beginserver_6;-1;'MacBook-Pro.local';socket/ip;0  
PDG_RESULT: mayaserver_begin_beginserver_6;-1;'59593';socket/port;0  
[User Log]: Nodes: [u'front', u'persp', u'side', u'top']
```

プロダクションの例



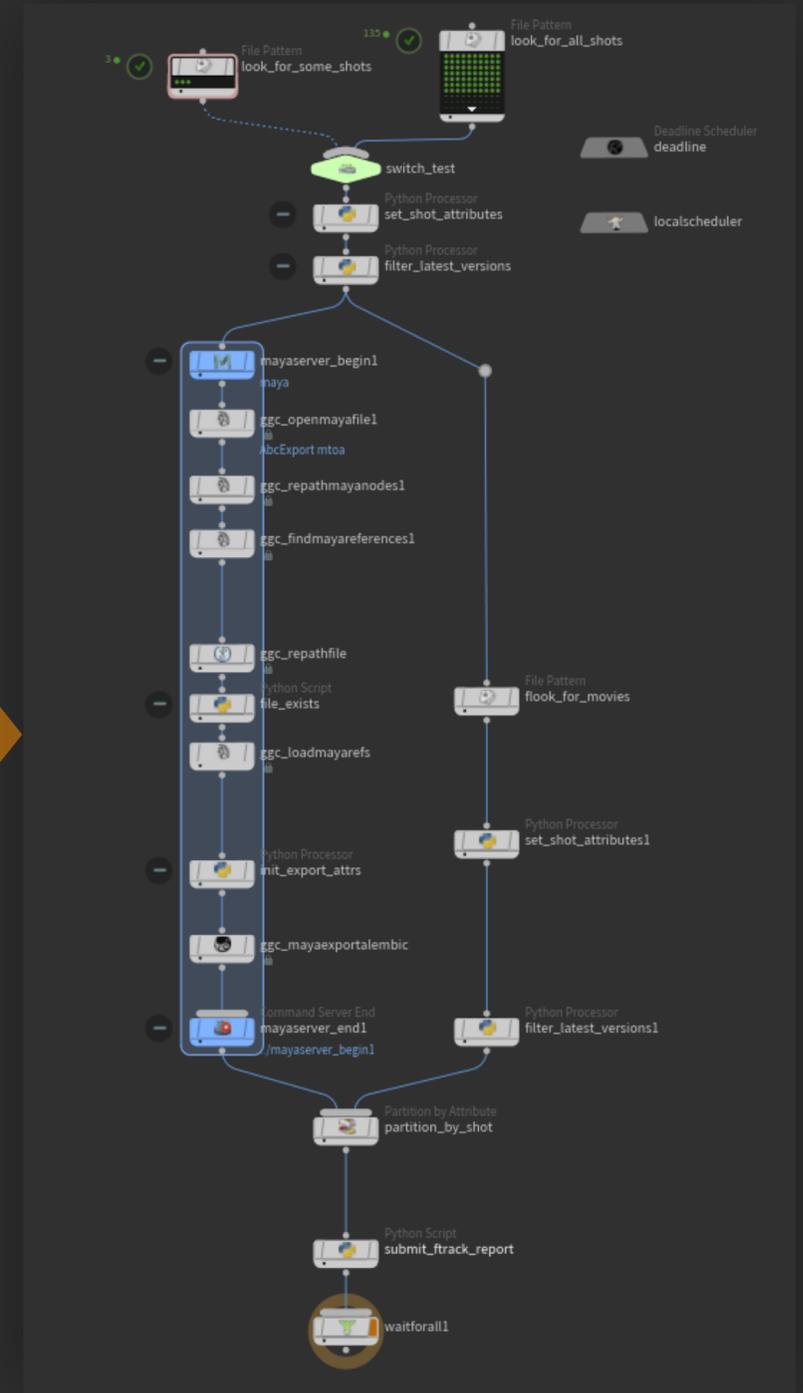
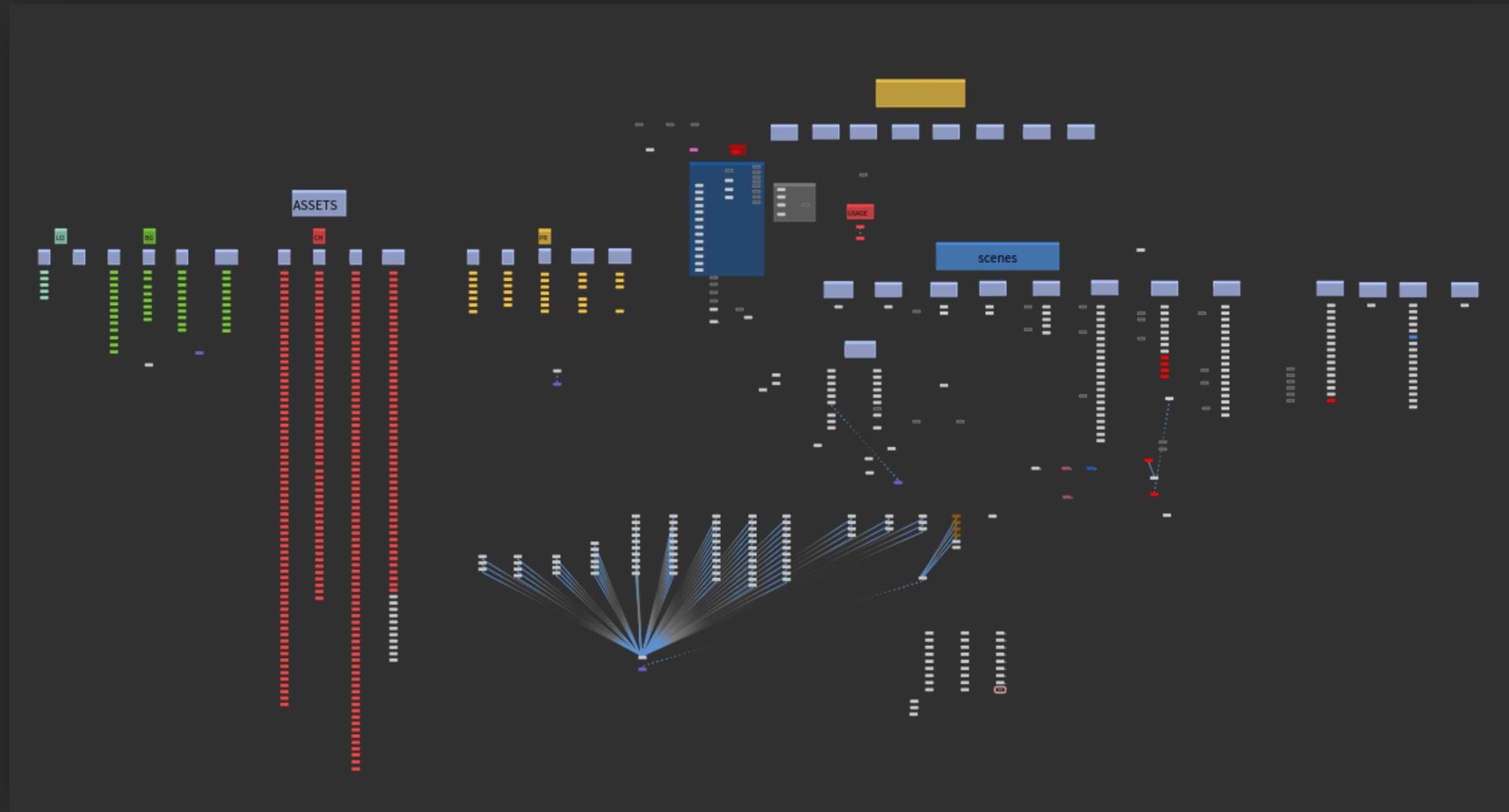
TOPベースパイプラインのメリット

- パイプラインジャンキーじゃなくても組めるようになってきた
- パイプラインのタスクをプロセスャル的に扱える
- Pythonのバケモノスクリプトを書かなくていい
- SOHOは覚えなくていい
- 開発とデバッグに社内のリソースを前ほどかけなくていい
- 覚えたら他のことにも使える

結論の代わり

TOPs

ROPs



ご静聴ありがとうございました。

Q&A