

# ClauJson

vztpv@naver.com

# Ideas

- 1. Divide checking grammar (a) and constructing ast. (b)
- 2. Use `std::thread` in (a), and (b)
- 3. (Total) Token Array. (simdjson`s stage1 as Tokenizer)
- 4. Divide Token Group, and Parallel Parsing (make partial ASTs)
- 5. Merge partial ASTs into one AST.

# Think Comma, and Colon.

- Suppose Valid Json String.
- Divide Token Array Using comma.
- , key : value or , value (] or , or value)

# Virtual Array, Virtual Object

- `, 1, 2 ] => [ 1, 2 ]`
- `, "abc" : true , "def" : 0 } => { "abc" : true, "def" : 0 }`

# Nested Virtual ~ ~

- 1, 2, 3 ] , 4, 5, 6, [ 7, 8, 9 ] ]  
=> [ [ 1, 2, 3 ] , 4, 5, 6, [ 7, 8, 9 ] ]
- Property?
  - [ [ [ ] ] ] exist
  - [ ] [ ] exist
  - [ ] [ ] not exist

# Partial Json

- Case `, 1, 2, 3`
- Case `, "key" : "value", "abc" : 1`
- Case `, 1, 2, 3 ], [ 4, 5, 6 => [ 1, 2, 3 ] , [ 4, 5, 6 ]`
- Partial Json Class has (Array's Elements, or Object's Elements), and (Virtual ~~).

# Parallel Parsing in each Thread.

- Suppose total token array is valid.
- No state variable.
- Use Stack or Node`s parent.
- Case , 1, 2, 3, ] , 4, 5, { "check" : [ 1, 2, 3  
=> [ 1, 2, 3 ] , 4 , 5 , { "check" : [ 1, 2, 3 ] }
- Root`s type is Partial Json. Meet {, now Node is {
- Meet ] or }, and Node is root, then make virtual ~~
- { "check" : [ 1, 2, 3 here Last Node`s Position.

# (Linear) Merge

- Using **Last Node's Position** and **Property of Virtual ~ ~**
- 1. Last Node's Position
- 2. repeat (Node is not NULL && first child of Node is virtual ~ ~)  
{  
    Node <- first child of Node.  
}
- 3. Now, x <- Node i, y <- Node i+1 `s Last Node's Position
- 4. Move Data from y to x, x <- x.parent, y <- y.parent.
- 5. And repeat 4. while ( x is not NULL && y is not NULL ) { ~ ~ }



